MASTER EN SCIENCES INFORMATIQUES

Mesures de complexité de programmes Pascal

Grégoire, Christophe

Award date:
1986

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES N.D DE LA PAIX
NAMUR

Institut d'informatique

Année académique 1985-1986

MESURES DE COMPLEXITE DE PROGRAMMES PASCAL

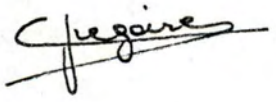
PROMOTEUR:

J.RAMAEKERS

Mémoire présenté
pour l'obtention
du grade de licencié
et maitre en
informatique par
GREGOIRE christophe

Je remercie vivement l'équipe de
Mr Van Lamsweerde et surtout Mr Ramaekers pour
les critiques, les renseignements et les conseils
apportés tout au long de ce travail.

Grégoire Christophe



PLAN

PLAN

PLAN

PLAN

PLAN

CHAPITRE 1:

INTRODUCTION

INTRODUCTION

De nos jours, peu d'informations sont disponibles sur les caractéristiques des programmes pendant les phases de développement et de maintenance et pourtant ce sont celles-ci qui demandent beaucoup de ressources et de temps. Il apparait donc important d'obtenir certaines mesures relatives à ces deux phases. On a vu ainsi apparaître à la fin des années 60 des métriques fournissant des paramètres interessants pour le programmeur, le concepteur, le directeur de projet, ... Ce mémoire va consister en l'étude des mesures faisant partie de la discipline appelée "Software science" une partie de "Software métrics".

Software Métrics est une jeune discipline faisant partie de la branche appelée Software Engineering. Son but est de mesurer certains aspects des softwares et de leurs développements. Ces métriques peuvent être utilisées pour estimer le coût d'un projet, le partage des ressources, pour évaluer l'efficacité d'une méthode de programmation, la durée d'un programme, le nombre d'erreurs dans les phases de tests, de maintenance, ... Elles sont donc utilisées dans chacune des phases de développement d'un programme.

Une "Software Métric" est une mesure du degré avec lequel un programme possède et exploite certaines qualités, propriétés ou attributs tels que maintenabilité, testabilité, flexibilité, coût, complexité, ... Beaucoup de nombres "magiques" peuvent être associés à ces caractéristiques: coéfficient de complexité, effort de programmation, ...

Deux grands courants sont apparus dès la création de cette discipline.

Le premier Software science étudie les mesures, les propriétés des algorithmes, leurs implémentations dans un langage. A l'intérieur de ce domaine, se distinguent deux approches différentes. La première est celle de Maurice Halstead[18] qui utilise le nombre d'opérandes et le nombre d'opérateurs comptés dans le programme. La deuxième approche est principalement l'oeuvre de Thomas McCabe. Celui-ci part de la représentation graphique du programme et en déduit une mesure de complexité.

Le deuxième courant est développé par Kenneth Kolence[23]. Il introduit une théorie appelée Software Physics pour l'étude de certaines caractéristiques d'exécution d'unité (programme ou collection de programmes). Parmis ces mesures se trouvent le travail, le temps d'exécution, le temps d'attente, l'espace mémoire occupé, ...

A partir de ce tois théories, d'autres modèles ont été développés par Gordon[16,17],Ottenstein[27], ...

Notre but étant ici d'étudier certaines mesures de programmes nous limiterons donc notre étude au premier courant:Software Science.

Le chapitre 2 est consistué d'un passage en revue de toutes les mètriques importantes, en commençant par celles de Halstead et McCabe.

Une fois que les principales mètriques auront été présentées, toutes les informations nécessaires à la création d'un outil de mesures sont réunies. Dans le chapitre 3 sera ainsi présenté un outil indispensable à la réalisation d'un automate rapide de calcul des complexités. Cet outil est l'analyseur syntaxique.

Cet outil présenté, le chapitre 4 explique les différentes phases nécessaires à la réalisation de l'automate. Les grandes lignes de l'algorithme sont présentées ainsi que les étapes secondaires indispensables.

Une application est développée dans le chapitre suivant. Deux versions différentes du programme des 8 reines sont étudiées, ainsi que toutes les transformations nécessaires pour passer de l'un à l'autre.

Le chapitre 6 contient l'analyse de certains domaines d'applications des mesures de complexité. Cette partie apporte des éléments quant aux informations indispensables aux programmeurs, aux chefs de projet, ...

Enfin, le chapitre 7 apporte une conclusion à cette étude.

CHAPITRE 2:

SOFTWARE METRICS

1. Introduction

Ce chapitre commence avec la présentation des deux principales métriques, celles de Halstead et McCabe, ainsi que les métriques de Gilb et Dijkstra. Ces dernières ne sont pratiquement jamais utilisées car elles sont fortement dépendantes du langage. Les deux premières sont par contre les plus anciennes et à la base d'un certain nombre d'autres développements.

La deuxième partie consiste en l'exposé de quelques utilisations de ces deux mesures avec en parallèle la présentation de quelques coéfficients de corrélation. Ces nombres expliquent le choix de telles métriques.

Après cette deuxième section, les extensions principales sont présentées. On parlera des mesures de Ottenstein, Gordon, Myers et d'autres.

2. Les premières mesures de complexité

2.1 Maurice Halstead

En 1972, M.Halstead commença à étudier certains algorithmes dans le but de tester une hypothèse qui lui apparut lors d'une compilation. Cette supposition était simplement: le nombre d'opérandes et d'opérateurs dans un programme sont en très grande corrélation avec le nombre de fautes découvertes dans ce dernier.

Ce nombre connu comme la longueur d'un programme fut comparé à d'autres "mesures" de complexité [14].
Les résultats furent surprenants puisqu'ils indiquaient que cette "longueur" était étrangement meilleure, comme approximation du nombre d'erreurs, que les différentes autres mesures .

Intrigué par ces détails, M.Halstead[18] postula en 1977 que les algorithmes pouvaient être caractérisés par un certain nombre d'invariants comme la longueur, le volume, le niveau de language, le niveau d'un programme, l'effort de programmation, ...

Examinons chacune de ces mesures.

Les mesures de Halstead sont donc entièrement basées sur quatre nombres. Ces derniers sont :

n_1 le nombre d'opérateurs distincts
n_2 le nombre d'opérandes distincts
N_1 le nombre total d'occurences des opérateurs
N_2 le nombre total d'occurences des opérandes

A partir de ceux-ci, il définit la longueur d'un programme N et la taille du vocabulaire n.

$$N = N_1 + N_2$$
$$n = n_1 + n_2 \qquad (2.1)$$

En partant de sa connaisance en thermodynamique, Halstead définit une estimation de la longeur.

$$N^* = n_1 \log_2 n_1 + n_2 \log_2 n_2 \qquad (2.2)$$

Il construit ensuite le volume V.

$$V = N \log_2 n = (N_1 + N_2) \log_2 (n_1 + n_2) \qquad (2.3)$$

L'intuition est simple: pour chacun des N éléments d'un programme, $\log_2 n$ bits doivent être spécifiés pour choisir un élément parmi les opérandes et les opérateurs. V représente donc le nombre de bits requis pour spécifier un programme.

Le volume potentiel où les opérations sont de simples appels de procédures est ensuite défini:

$$V^* = (n_1^* + n_2^*) \log_2 (n_1^* + n_2^*) \qquad (2.4)$$

au minimum: $n_1^* = 2$, $N_1^* = n_1^*$ et $N_2^* = n_2^*$.

Halstead émet l'hypothèse qu'une loi de conservation entre le niveau d'abstraction et le volume V existe. Cela se traduit par la relation suivante:

$$L\ V = \text{constante} \qquad (2.5)$$

Le niveau L est ainsi défini comme le quotient du volume potentiel sur le volume.

$$L = \frac{V^*}{V} \qquad (2.6)$$

Halstead argumente alors que le niveau du programme peut augmenter avec le nombre d'opérandes distincts employés et diminuer avec le nombre d'opérateurs distincts et le nombre d'occurences d'opérandes. Ce qui l'amène à proposer une estimation du niveau d'un programme:

$$L^* = \frac{2\, n_2}{n_1\, N_2} \qquad (2.7)$$

Halstead suggère ensuite que le produit du niveau du langage par le volume potentiel est une constante nommée lamda et appelée niveau du langage.

$$\text{lamda} = L\, V^* \qquad (2.8)$$

Le niveau du langage peut encore s'écrire de la façon suivante si on utilise la relation (2.6):

$$\text{lamda} = L^2\, V \qquad (2.9)$$

Comme la difficulté de programmer augmente si le volume augmente et décroit si le niveau du programme augmente, il propose le nombre E comme mesure de l'effort mental requis pour créer un programme.

$$E = \frac{V}{L} \qquad (2.10)$$

L'effort mental est aussi appelé effort de programmation ou effort de lisibilité. Cette définition a pour conséquence le fait suivant: si le niveau du langage utilisé est grand, l'effort mental diminue. Celui-ci peut, en partant des relations précédantes encore s'écrire:

$$E = \frac{(V^*)^3}{lamda^2} \qquad (2.11)$$

Halstead émet ensuite l'hypothèse que le temps de programmation T doit être proportionnel à l'effort de programmation.

$$T = \frac{E}{S} \qquad (2.12)$$

où S est une constante.

La constante S représente la vitesse du programmeur, c'est à dire le nombre de discriminations mentales par seconde dont il est capable. Il prend S=18 comme valeur raisonnable et établit une relation avec le nombre de Stroud. En effet, Stroud établit que le nombre de discriminations mentales est compris entre 5 et 20 par seconde. La valeur choisie par Halstead est donc très proche de la borne supérieure de Stroud, c'est pourquoi Halstead appelle S le nombre de Stroud. T peut aussi être utilisé pour estimer le temps nécessaire à un programmeur pour lire et comprendre un programme .

En partant de certains tests réalisés, il se rend compte que l'effort E est le meilleur prédicateur du nombre d'erreurs. Ce résultat lui suggère une étrange relation entre le nombre de discriminations mentales pour créer un programme et le nombre d'erreurs commises en programmant. Pour qualifier cette relation, Halstead définit le nombre d'erreurs B comme étant le quotient de l'effort de programmation sur le nombre moyen E_o de discriminations entre deux erreurs.

$$B = \frac{E}{E_o} \qquad (2.13)$$

Il construit ensuite une estimation de ce nombre d'erreurs.

$$B^* = \frac{K\ V}{E_o} \qquad (2.14)$$

Ce nombre B représente le nombre d'erreurs faites en créant un module. Il ne peut pas être utilisé pour estimer la fin des tests, c'est à dire essayer d'obtenir B=0. La localisation et suppression des erreurs ne réduissent pas B à 0.

2.2 Thomas McCabe

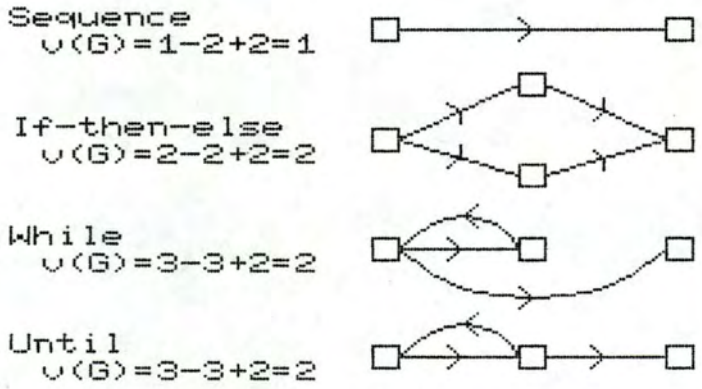
Le point de départ de McCabe[24] en 1976 est totalement différent puisqu'il part d'une représentation graphique G du programme où chaque noeud correspond à un bloc de codes et où chaque arc correspond à une alternative du programme. Une méthode pour calculer la complexité serait de mesurer le nombre de chemins à travers un graphe. Mais si le programme possède un retour vers l'arrière , le nombre de chemins dans le graphe correspondant sera infini. C'est pourquoi McCabe définit la complexité d'un programme comme le nombre cyclomatique v(G) du graphe associé G.

$$v(G) = e - n + 2 \qquad (2.15)$$

où e est le nombre d'arcs et n le nombre de sommets.

Les exemples du graphe 2.1 illustrent cette définition.

Graphe 2.1



Après plusieurs expériences, McCabe conclut que le nombre cyclomatique v(G) est aussi le nombre de points de décision dans le programme plus un ou encore le nombre de régions délimitées par le graphe.

2.3 Gilb

Gilb[15] dans son livre "Software Metrics" énonce que la complexité est une mesure du degré de décision à l'intérieur d'un système;le nombre de if en est donc une bonne mesure. Plusieurs tests ont montré que cette mesure était une bonne estimation du coût d'un software.

2.4 Dijkstra

Dans sa lettre intitulée "Goto statement considered harmful", Dijkstra[7] observe que la qualité d'un programmeur est inversément proportionnelle à la densité des goto dans le programme. Cela suggère que le nombre de goto est une simple mesure de la complexité.

3. Quelques utilisations

Plusieurs tests et mesures sur de nombreux programmes ont été réalisés pour essayer de montrer que les différentes mesures proposées par Halstead et McCabe étaient représentatrices de la complexité d'un programme, du nombre d'erreurs, du temps de programmation, ... La plupart des expériences sont des études statistiques. Elles comparent les métriques de Halstead et McCabe. Elles comparent également les nombres obtenus par mesure et les nombres correspondants obtenus par calcul. La justesse des mesures est exprimée par les coéfficients de corrélation entre mesures et estimations.

Présentons en quelques unes.

3.1 Curtis,Sheppard,Borse,Love et Milliman

Parmi les mesures de corrélation entre les coéfficients de complexité citons celles réalisées par les auteurs qui suivent: Curtis, Sheppard, Borse, Love et Milliman[5, 6]. Ces trois auteurs ont réalisé des mesures sur des versions différentes d'un même programme et des procédures de celui-ci. Ils ont ainsi pu parvenir aux intercorrélations du tableau 2.1.

Tableau 2.1

mesures	Corrélations E	v(G)
procédure:		
v(G)	.92	
longueur	.89	.81
programme:		
v(G)	.76	
longueur	.56	.90

On voit donc au niveau procédure que les trois mesures ont des performances plus ou moins égales. Au niveau programme, il n'en va pas de même puisque les mesures de corrélation avec E sont plus faibles. En partant des mêmes mesures, ils établissent encore les corrélations entre performances et complexités. Les résultats sont présentés dans le tableau 2.2.

Tableau 2.2

mesures	Corrélations routines	programme
E	.66	.75
v(G)	.63	.65
longueur	.67	.52

Pour les procédures, les mesures donnent des performances similaires. Pour le programme, d'étranges différences sont observées dans la capacité qu'ont les mesures de complexité de prédire la performance.

Ces auteurs apportent les conclusions suivantes. Premièrement, la grandeur des programmes étudiés peut être trop restrictive pour observer la taille des relations, ce qui peut entrainer des résultats erronnés. En effet, les programmes étudiés ne dépassent pas 55 lignes. Deuxièmement, il apparaît que les mesures de Halstead et McCabe sont en relation avec l'expérience difficile des programmeurs à détecter les erreurs. Mais ces mesures sont capables de satisfaire d'autres exigences comme fournir un feedback sur la complexité d'un programme, donner un ordre de grandeur du temps nécessaire pour comprendre un programme, ... Cette seconde partie sera étudiée en détail au chapitre 6. Troisièmement, l'ordre des mesures de complexité est le suivant:en tête Halstead, suivi par celle de McCabe et enfin par la longeur d'un programme.

3.2 Fitzimmons et Love

Fitzimmons et Love [14] rapportent d'autres coéfficients de corrélation. Les mesures ne portent plus sur la comparaison des complexités mais sur d'autres éléments. Ils étudient les relations entre mesures

observées et mesures calculées, entre la complexité de Halstead et divers paramètres. Certaines relations sont données dans le tableau 2.3.

Tableau 2.3

Les variables		Corrélation
. nombre de fautes	complexité E	.98
. nombre de fautes	nombre de décisions plus les appels	.83
. temps moyen de debugging	complexité E	.20
. longueur observée	longueur calculée	.95
. temps de programmation observé	temps de programmation calculé	.93

Il apparait suite à ces mesures de corrélation que la complexité de Halstead est assez représentative du nombre d'erreurs, mais ne l'est pas du temps moyen de debugging et que la longueur calculée et le temps calculé sont de la même grandeur que longueur mesurée et temps mesuré.

Certains de ces tests montrent aussi que les deux mesures de complexité sont de bonnes estimations des tâches de maintenance, des performances des programmes et des systèmes.

3.3 Weissman

Il apparait dans certaines de ces mesures que les coéfficients de complexité dépendent de certaines impuretés. Weissman[29] présente un nombre considérable d'éléments qui influencent la complexité. Il les regroupe en quatre classes: forme de programme, flux de contrôle, flux des données et interaction entre les deux flux. La première classe contient les facteurs qui affectent l'apparence et la lisibilité du programme. Les trois classes restantes sont constituées des facteurs qui

influencent la structure du programme. Parmi ces classes, on retrouve des éléments comme:le choix des noms de variables, le placement des déclarations, l'utilisation de noms non significatifs, l'utilisation de pointeurs, la longueur des segments, ...

3.4 Elshoff

Il faut aussi citer Elshoff[10,11,12,13]. Il effectue un certain nombre de mesures relatives à des algorithmes écrits en PL/I suivant cinq critères:la taille du programme, la lisibilité de celui-ci, d'un langage de programmation. Il regroupe ses résultats dans des tableaux. Il détermine aussi la justesse du choix des quatre variables de Halstead n_1, n_2, N_1, N_2 et compare plusieurs méthodes d'estimation du nombre d'opérateurs et du nombre d'opérandes. Il se rend compte que la plupart des programmes mesurés sont trop grands, sont souvent illisibles et sont extrèmement complexes. De plus, souvent, les programmeurs ne connaissent pas suffisamment, le langage utilisé.

4. Les améliorations

Dans cette section seront présentées des améliorations et de nouvelles mesures construites sur celles de McCabe et Halstead.

4.1 Baker et Zweben

Baker et Zweben[1, 2] se sont rendus compte que la structuration et la modularité des programmes influencaient la complexité, c'est pourquoi ils ont développé une théorie améliorant celle de Halstead en introduisant la modularité.

Elle est, pour les auteurs,le résultat d'une approche particulière du programme à résoudre et est exprimée en terme de sous-unités qui sont développées indépendamment et sont ensuite reliées entre elles. La modularité est achevée en isolant les parties de code similaire. Ils distinguent deux groupes de transformations: les sous-expressions communes et les sous-séquences communes.

Examinons chacune des transformations.

Soient n_1, n_2, N_1, N_2, N et V relatives à la première version et n_1', n_2', N_1', N_2', N' et V' relatives à la seconde version.

a) Transformation des sous-expressions

Si plusieurs expressions identiques sont détectées, la première occurence de la sous-expression est suivie par un assignation de sa valeur à un nouvel opérande et ensuite chaque occurence de la sous-expression est remplacée par ce nouvel opérande. Le "code" ainsi obtenu est équivalent au "code" non modularisé.

Posons j le nombre de répétitions de la sous-expression, ce qui correspond à j+1 occurences de celle-ci et m la longueur de cette sous-expression.
On a alors en fonction des anciennes mesures de Halstead:

$$n_1' = n_1$$

$$n_2' = n_2 + 1 \quad \text{par ajout du nouvel opérande}$$

$$N' = N + (m+3) - (j+1)(m-1)$$

Cette formule est obtenue par ajout des occurences du nouvel opérande, l'opérateur d'assignement, la sous-expression de longueur m et l'opérateur de fin et soustraction de (j+1) occurences de la sous-expression remplacée par un opérande

Par substitution, le volume devient:

$$V' = (N + 4 - j(m-1)) \log_2(n+1) \qquad (2.16)$$

Il s'en suit que le nouveau volume est inférieur à l'ancien si j vérifie la condition suivante:

$$j > \frac{N + 4 - N \log_2(n)/\log_2(n+1)}{m - 1} . \qquad (2.17)$$

La réduction du volume est donc dépendante du nombre n_1+n_2, de la séquence originale, du nombre de caractères dans celle-ci et de sa longueur. Elle est interessante si le nombre de répétitions est important.

b) Transformation de sous-séquences

Lorsque une telle sous-séquence est rencontrée, il est possible d'implémenter celle-ci sous la forme d'une procédure et à chacune de ces rencontres, d'appeler la sous-routine.

Soient j répétitions de la sous-séquence de longueur m, p opérandes passant en paramètres et q opérandes locales. On obtient alors:

$n_1' = n_1 + 1$ à cause de l'invocation

$n_2' = n_2 + q + p$

$N' = N + m - (j+1)(m-2p-2)$

La longueur est obtenue par ajout de m appels de procédure, et chacune des j+1 occurences de la procédure

est remplacée par un opérateur identificateur, un opérateur de groupement des arguments, p arguments, p-1 séparateurs et l'opérateur de fin.

Le volume devient:

$$V' = (N + m - (j+1)(m-2p-2)) \log_2(n+q+p+1) \quad . \qquad (2.18)$$

De nouveau, le volume est inférieur à l'ancien si j vérifie la propriété suivante:

$$j > \frac{N + 2p + 2 - N\log_2(n)/\log_2(n+p+q+1)}{m - 2p - 2} \quad . \qquad (2.19)$$

La réduction du volume est donc fonction du nombre n_1+n_2 du programme, le nombre d'appels, la longueur de la sous-séquence, le nombre d'opérandes passant en paramètres et le nombre d'opérandes locaux. De nouveau, la diminution n'est intéressante que si le nombre de répétitions est suffisamment grand.

4.2 Gordon

Gordon[16,17] en étudiant Halstead se rend compte que les six classes d'impuretés proposées par ce dernier influencent l'effort de programmation. Ces six classes sont les suivantes:opérations complémentaires, opérandes ambigus, opérandes synonymes, sous-expressions communes, assignements non nécessaires et expressions non factorisées. Chaque classe est caractérisée par un exemple du tableau 2.4.

Tableau 2.4

impuretés	solutions
opération complémantaire:	
P+Q→T;T*T+T-T→R	P+Q→T;T*T→R
opérande ambigu:	
P+Q→R;R*R→R	P+Q→T;T*T→R
opérande synonyme:	
P+Q→T1;P+Q→T2;T1*T2→R	P+Q→T1;T1*T1→R
sous-expression commune:	
P*(P+Q)+Q*(P+Q)→R	P+Q→T;P*T+Q*T→R
assignement non nécessaire:	
P+Q→T1;T1^2→R	(P+Q)^2→R
expression non factorisée:	
P*P+2*P*Q+Q*Q→R	(P+Q)^2→R

Partant des ces classes, Gordon définit une mesure appelée clareté d'un programme, notée E_c pure.

Pour réduire l'influence de ces impuretés sur la mesure de l'effort de programmation, il part de la mesure de Halstead estimée

$$E_p = \frac{N^* \log_2 n}{L^*} \qquad (2.20)$$

ou encore

$$E_c = \frac{V}{L^*} \quad . \qquad (2.21)$$

On obtient donc par substitution l'effort de programmation impure:

$$E_c \text{ impure} = \frac{(N_1 + N_2)\, n_1\, N_2 \log_2(n_1 + n_2)}{2\, n_2} \quad . \qquad (2.22)$$

Après avoir donné une mesure de l'effort mental avec impuretés, il distingue ensuite chacune des classes

d'impuretés et propose pour chacune une mesure de la clareté du programme dans lequel les impuretés ont été supprimées.

Examinons chacune des classes.

a) opérateurs complémentaires

Ces opérations s'annulant l'une l'autre, il est évident qu'elles peuvent être supprimées. La mesure de l'effort purifiée de ces opérations devient:

$$E_c \text{ pure} = \frac{(N_1 + N_2 - 2)\, n_1\, N_2 \log_2(n_1 + n_2)}{2\, n_2} \qquad (2.23)$$

en effet la purification réduit le nombre total d'occurences d'opérateurs par 2.

b) opérandes ambigus

Dans certaines situations, il est possible d'utiliser la même variable pour représenter deux types différents dans deux portions différentes du programme. Mais lorsque ces deux types sont dans la même portion, cela devient ambigu. Gordon propose la mesure suivante:

$$E_c \text{ pure} = \frac{(N_1 + N_2)\, n_1\, N_2 \log_2(n_1 + n_2 + 1)}{2\,(n_2 + 1)} \qquad (2.24)$$

en effet on introduit un opérande unique et donc n_2 augmente d'une unité.

c) opérandes synonymes

A la place d'utiliser un nom pour deux quantités différentes, on peut employer deux noms pour la même

Cela représente une impureté qui peut être supprimée:

$$E_c \text{ pure} = \frac{(N_1 - N_1' + N_2 - N_2')\ n_1\ (N_2 - N_2')\ \log_2(n_1 + n_2 - 1)}{2\ (n_2 - 1))} \quad . \qquad (2.25)$$

Une telle expression concerne N_1' opérateurs et N_2' opérandes et donc après purification, N_1 et N_2 sont réduit respectivement de N_1' et N_2' et n_2 est diminué d'une unité car le synonyme est remplacé.

d) sous-expression commune

Lorsqu'une sous-expression commune est découverte, il est préférable de lui assigner un nouveau nom et d'utiliser ce nouveau nom dans chacune des occurences de cette expression. Cela mène à l'effort suivant où n est le nombre d'occurences:

$$E_c \text{ pure} = \frac{(N_1 - N_1' + N_2 - N_2' + 5)\ n_1\ (N_2 - N_2' + 3)\ \log_2(n_1 + n_2 + 1)}{2\ (n_2 + 1))} \qquad (2.26)$$

pour n=2, car n_2 augmente d'une unité, N_1 augmente de 2 (l'opérateur d'assignation et l'opérateur de fin), enfin N_1 décroit de $(n-1)N_1'$ et N_2 décroit de $(n-1)N_2'$.

e) assignations non autosisées

Ce type d'impureté provient de l'évaluation d'une expression, de l'assignation de celle-ci à une variable unique et ensuite de l'utilisation de cette variable. Celui-ci est supprimé en enlevant l'affectation du programme. Cela donne:

$$E_c \text{ pure} = \frac{(N_1 + N_2 - 4)\ n_1\ (N_2 - 2)\ \log_2(n_1 + n_2 - 1)}{2\ (n_2 - 1)} \quad . \qquad (2.27)$$

N_1 et N_2 diminuent de 2 unités et n_2 décroit d'une unité.

f) expressions non factorisées

Il est clair qu'une expression qui peut être factorisée est plus facile à comprendre lorsque la factorisation a été réalisée. Celle-ci introduit de nouveaux opérateurs. Pour l'exemple ci-dessus, la formule de l'effort devient:

$$E_c \text{ pure} = \frac{(N_1+N_2-7)\ n_1\ (N_2-4)\ \log_2(n_1+n_2+2)}{2\ n_2} . \qquad (2.28)$$

Conclusion:

Il faut noter qu'il n'existe pas toujours une méthode pour découvrir et supprimer les impuretés. Mais lorsque celles-ci ont été découvertes et corrigées, l'effort nécessaire à la compréhension, c'est à dire l'effort de programmation revu par Gordon, est inférieur à celui donné par Halstead. Ces types d'impuretés se retrouvent en général chez les programmeurs non expérimentés, mais quelques programmes étudiés par Gordon, écrits par des personnes expérimentées, en possèdent quelques-unes. Ce type d'erreurs n'est donc pas, pour Gordon, limité aux programmeurs inexpérimentés.

4.3 Ottenstein

Otenstein[27] présente un modéle destiné à estimer le nombre d'erreurs restant dans le système au commencement des tests et des phases d'intégration. Ce modèle est basé sur les mesures de Halstead.

Le nombre d'erreurs introduit par Halstead est donné

par

$$B = \frac{E}{E_o} \qquad (2.29)$$

où E_o est le nombre moyen de discriminations mentales entre les erreurs.

Ottenstein propose une mesure du nombre d'erreurs en fonction du niveau du programme

$$B_{\checkmark}^{*} = \frac{L\ E}{E_o} \qquad (2.30)$$

ou encore en utilisant la relation (2.10):

$$B_{\checkmark}^{*} = \frac{V}{E_o} \ . \qquad (2.31)$$

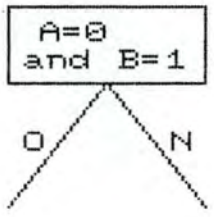
Elle présente aussi un certain nombre d'extensions de ce modèle. Parmi celles-ci, on peut citer:estimation du temps de détection et de correction d'une faute.

4.4 Myers

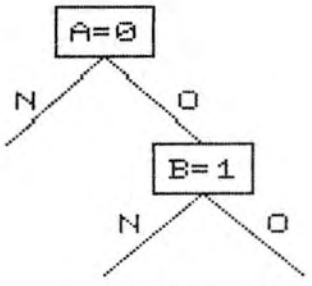
En étudiant la mesure de complexité de McCabe, Myers[26] s'est rendu compte que certaines anomalies existaient. Parmis celles-ci, on trouvait une mesure de complexité plus élevée pour un programme simple que pour un programme complexe. Il propose pour y remédier une extension du nombre cyclomatique de McCabe. Celui-ci était calculé sur la représentation graphique du programme. Mais il y a deux façons de dessiner un diagramme. L'exemple du graphe 2.2 le prouve.

Graphe 2.2

IF (A=0) and (B=1) THEN....



v(G)=2



v(G)=3

Si on considère les exemples suivants, on se rend compte d'une autre anomalie.

```
A: if (X=0) then ...
           else ...

B: if (X=0) and (Y>1) then ...
                      else ...

C: if (X=0) then
            if (Y>1) then ...
                     else ...
           else ...
```

Tout lecteur se rend compte que C est plus complexe que B qui est lui même plus compliqué que A, donc A<B<C. Si on calcule maintenant la complexité v(G), on obtient v(A)=2, v(B)=3, v(C)=3. Et donc 2<3<3 est faux.

Il propose de prendre un intervalle de complexité v(G)=a:b où a est le nombre de décisions plus un et b est le nombre de conditions plus un (un "case" à n "branches" est

compté comme n-1 conditions). Avec cette nouvelle mesure, on obtient v(A)=2:2, v(B)=2:3, v(C)=3:3, ce qui satisfait bien la relation implicite définie plus haut.

Cet intervalle se note aussi CYCMID:v(G).

4.5 Hansen

Pour Hansen[19], une bonne mesure de complexité doit satisfaire plusieurs critères. Ceux-ci sont:
- être en relation avec la complexité psychologique du programme
- servir d'outil de comparaison de plusieurs versions d'un programme
- donner la préférence à un programme sans instructions pauvres
- être indépendant d'autres mesures.

La définition du nombre cyclomatique est le nombre de branches dans le programme plus un. Le nombre de branches peut être, selon Hansen, calculé de plusieurs façons:
- a) le nombre de "if", "case" ou autre construction alternative
- b) le nombre de "do", "do-while" ou autre construction répétitive
- c) le nombre d'alternative dans un "case" moins deux
- d) chaque opérateur logique dans un "if".

McCabe utilise les quatre définitions. Myers étend cela à un intervalle a:b où a utilise les trois premières définitions et b est la mesure définie par McCabe. Hansen propose une mesure sous forme de couple (c, d) où c est une mesure du nombre cyclomatique utilisant les deux premières définitions et d est une mesure du nombre d'opérations. Cette mesure est encore appelée CYCMIN:N_1. Les exemples suivants donnent une idée de cette mesure.

```
*  If (X=0) then ...                          v(G) = (2, 1)
   else... .
*  If (X=0) and (Y>1) then ...                v(G) = (2, 2)
   else...
*  If (X=0) then
          If (Y>1) then ...                   v(G) = (3, 2)
          else... .
   else ... .
```

4.6 Ejiogu

Partant d'une représentation graphique où les noeuds représentent les blocs de données et les arcs donnent le cheminement entre ces blocs, Ejiogu[9] définit la complexité en se servant des trois attributs suivants d'un graphe:

- le niveau L :le nombre maximum de niveaux dans le graphe
- le nombre d'explosions E :le nombre de noeuds du premier niveau
- la monadicité M :le nombre de feuilles terminales dans l'arbre si L ≠ 1 et 1 sinon.

Il définit ensuite l'image de complexité

$$C = L\ E\ M \qquad (2.32)$$

et le degré de complexité

$$D_c = \frac{1}{C} . \qquad (2.33)$$

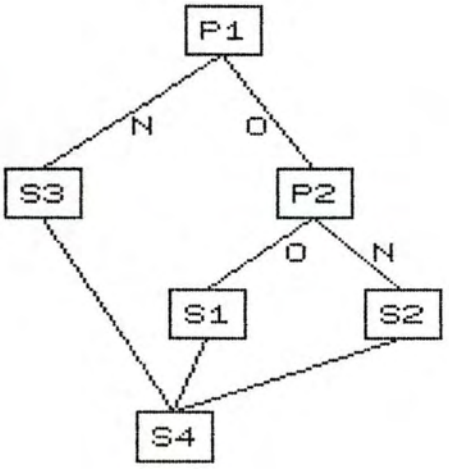
4.7 Harrison et Magel

Harrison et Magel[20,21] définissent une mesure basée sur le graphe de contrôle qu'ils appellent SCOPE RATIO ou rapport étendu. Celui-ci se définit comme le quotient du nombre de noeuds sur le nombre appelé SCOPE NUMBER. Ce dernier est la somme de la complexité de chaque noeud du graphe de contrôle. Pour calculer cette complexité, les auteurs distinguent deux sortes de noeuds:les noeuds attributs et les autres. La complexité des noeuds de la deuxième classe est égale à la complexité du noeud. Pour l'autre classe, cela est plus difficile. Pour chacun de ces noeuds, il faut construire le sous-graphe H qui est constitué des noeuds situés à "portée" de celui considéré. La complexité d'un noeud est alors calculée comme la somme des complexités de chaque noeud de H et de la complexité du noeud lui-même. Par exemple:

```
if P1 then
      if P2 then S1;
            else S2;
      else S3;
S4;
```

L'exemple est représentable par le graphe 2.3.

Graphe 2.3



Le sous-graphe de P1 est constitué par S3, P2, S1 et S2. Celui de P2 est formé de S1 et S2.

5. Conclusions

L'étude de la complexité remonte déjà à quelques années mais jusqu'à présent seul un petit nombre de métriques ont été construites.

Les premières métriques sont l'oeuvre de Halstead et McCabe. Un certain nombre d'auteurs les ont utilisées pour mesurer des programmes. Ils ont montré les relations qui existaient entre ces métriques et un certain nombre de

paramètres comme la longueur d'un programme, le nombre d'erreurs,...

Sur ces deux mesures de complexité sont venues se greffer toute une série d'autres métriques. Ces dernières répondent à de petites imperfections dans les définitions de Halstead et McCabe et représentent un développement plus poussé de celles-ci.

CHAPITRE 3:

L'ANALYSEUR SYNTAXIQUE

L'ANALYSEUR SYNTAXIQUE

1. Introduction

Après avoir présenté un certain nombre de métriques, il nous faut maintenant étudier comment les réaliser. Pour se faire nous allons limiter notre étude à la mesure de complexité donnée par Halstead ainsi que celle construite par de McCabe. Ces différentes mesures ont été choisies, parce que suivant les études présentées au chapitre précédent, elles se rapprochent le plus de la mesure de complexité que l'on veut étudier.

Plusieurs solutions sont possibles.

Une première solution consiste à partir du listing source du programme, à calculer manuellement les quatre nombres de Halstead ainsi que le nombre de points de décision. Cette manière de procéder peut être longue et fastidieuse si le programme est important et les comptes doivent être menés avec discipline.

Une autre solution consiste à réaliser un outil automatique de mesure. Cela va du simple programme analysant, caractère par caractère le programme texte à l'analyseur syntaxique. Le premier outil peut prendre un certain temps si le programme source possède un important nombre de caractères. Il serait nécessaire de créer une table contenant tous les mots réservés du language dans lequel le programme est écrit. Le deuxième outil est plus rapide et fournit de plus un contrôle syntaxique du programme source. Ce dernier a retenu notre attention.

2. Fonctionnement de l'analyseur syntaxique

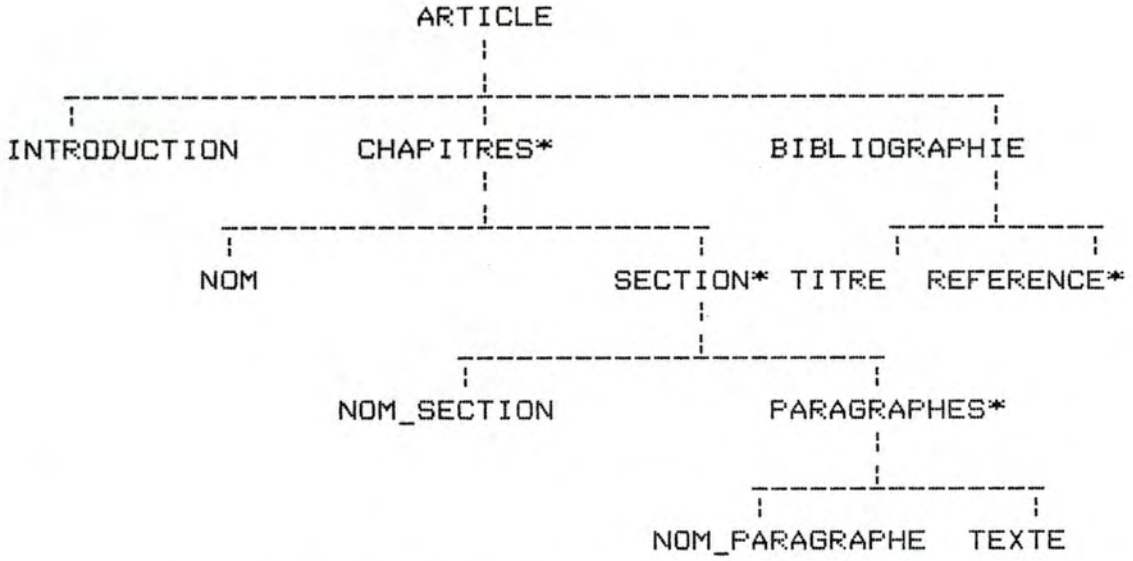
2.1 Structure d'un programme

Considérons un article d'un auteur quelconque, ce texte possède une certaine structure. En effet, il peut être décomposé en une introduction, en un corps et en une bibliographie. Le corps peut lui-même être découpé en un certain nombre de chapitres, eux-mêmes décomposés en paragraphes. Une définition de texte structuré est la suivante:

Un texte structuré est un texte qui, bien qu'apparaissant extérieurement sous la forme d'une suite de caractères, présente une structure profonde en terme de composants et de relations entre composants, reflétant un concept syntaxico-sémantique.

Ces concepts syntaxico-sémantiques correspondent généralement à des entités syntaxiques intervenant dans la définition formelle d'une syntaxe. Cette structure se représente facilement sous forme d'un arbre (graphe 3.1).

Graphe 3.1



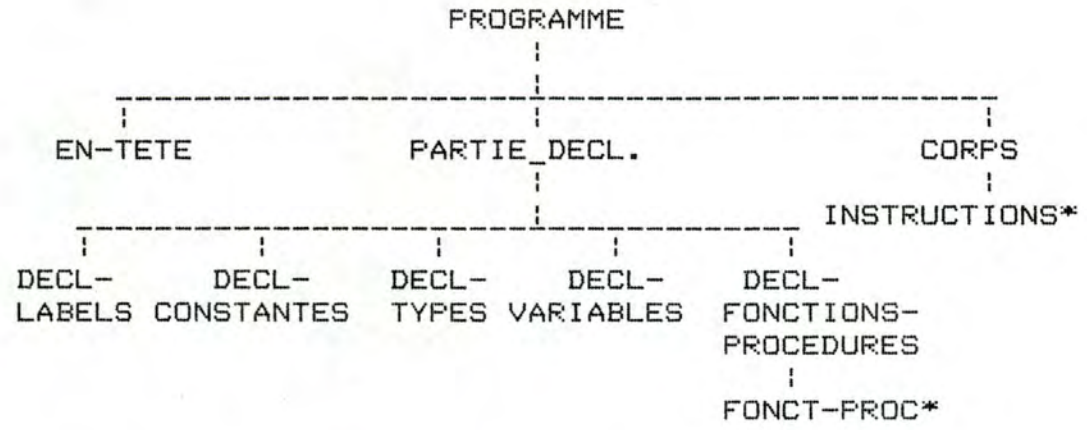
Les programmes sources peuvent de la même façon être structurés sous forme arborescente (graphe 3.2).

Un programme comprend un en-tête, une partie déclarative et un corps.

La partie déclaration peut contenir une partie déclaration des labels, une partie déclaration des constantes, une partie déclaration des types, une partie déclaration des variables et enfin la partie des déclarations des fonctions et procédures. Chacunes des parties déclaratives peuvent être décomposées en sous parties. Par exemple, la partie déclaration de fonctions et procédures comprend un certain nombre de fonctions ou procédures.

Le corps du programme est constitué d'instructions, celles-ci pouvant, elles aussi, être décomposées.

Graphe 3.2



Cette structuration est rendue possible grâce à l'utilisation d'une syntaxe.

2.2 Définitions des syntaxes

Un langage possède deux syntaxes: l'une concrète et l'autre abstraite. Donnons pour chacune une définition.

2.2.1 La syntaxe concrète

La syntaxe concrète permet de déterminer les constructions syntaxiques admises dans le langage. Elle se présente généralement sous forme d'un ensemble de règles permettant de déterminer si une suite de caractères est l'expression d'un concept dans le formalisme du langage.

Cette syntaxe concrète est, par exemple, exprimée en B.N.F. Pour représenter un article, la syntaxe sera la suivante dans ce symbolime:

```
<article> ::= INTRODUCTION <intro> <chapitre>* <bibliographie>
<chapitre> ::= CHAPITRE <int> <nom_chapitre> <txt_chapitre>
<txt_chapitre> ::= SECTION <int> <nom_section> <txt_par>
<txt_par> ::= par <nom_par> <txt>.
```

Un langage se traduit par une suite de règles. Le membre de gauche est l'entité syntaxique définie par la règle, tandis que la partie de droite représente la définition de cette entité. Celle-ci peut être soit une composition d'entités définissantes, soit une définition générique.

La composition d'entités définissantes peut être séquentielle, sélective ou itérative et est agrémentée de mots réservés.

Exemples de composition :

```
<sequentielle>  : <if> ::= if <cond> then <stat> else <stat>
<selective>     : <instr> ::= <if>|<while>|<for>|<assignation>
<iterative>     : <liste> ::= <elem>*.
```

Une définition générique d'une entité syntaxique revient à définir par une expression régulière les suites de caractères valables pour toute occurence de cette entité.

Exemple:

```
<identificateur> ::= [a_z]*.
```

2.2.2 La syntaxe abstraite

La syntaxe abstraite est la syntaxe concrète de laquelle les éléments lexicaux, les mots et symbôles réservés sont absents. Elle est utilisée pour définir la structure de la représentation interne. On l'appelle parfois la syntaxe des arbres.

Exemple:

```
<if> ::= <cond> <stat> <stat>.
```

2.3 Arbre abstrait

Comme nous l'avons vu avant, chaque texte possède une représentation interne appelée arbre abstrait.

Dans cet arbre, chaque noeud représente une entité syntaxique du langage, c'est à dire la partie gauche des règles de syntaxe concrète, et les arcs orientés d'un noeud père vers des noeuds fils représentent la relation de définition entre entités définies et entités définissantes.

Les feuilles de l'arbre sont les génériques du langage ou les noeuds n'ayant pas de fils.

2.4 Passage de la représentation externe à la représentation interne

Le passage d'une forme externe à une représentation interne est réalisée grâce à un automate. Celui-ci utilise un analyseur syntaxique, qui lui-même utilise un analyseur lexical.

L'entrée de l'automate est la représentation textuelle de l'objet à analyser. La sortie correspond à la représentation arborescente de ce texte.

La gestion de ce texte en entrée est confiée à l'analyseur lexical. Celui-ci a pour rôle la découpe du texte en tokens, c'est à dire en unités lexicales du formalisme. Ces unités sont les mots et symboles réservés et les unités syntaxiques définies par des expressions régulières.

Exemple d'analyse lexicale:

Pour le texte pascal suivant:

```
function calcul_énergie (m:real;c:real):real;
begin
   calcul_énergie := m*c**2;
end;
```

l'analyse lexicale (les tokens distingués sont marqués en les soulignant) donne:

```
function calcul_énergie ( m : real ; c : real ) : real ;
begin
   calcul_énergie := m * c ** 2 ;
end ;
```

L'analyseur lexical va indiquer à l'analyseur syntaxique les éléments rencontrés ainsi que leur catégorie.

Les informations communiquées pour le début de l'exemple ci-dessus sont données dans le tableau 3.1.

Tableau 3.1

TYPE	VALEUR
mot_function	
identificateur	calcul_énergie
parenthèse ouvrante	
identificateur	m
deux point	
type	real
....	

L'analyse syntaxique consiste alors à vérifier que la suite des tokens identifiés par l'analyseur lexical vérifie bien la syntaxe du formalisme donné. Il effectue également

des appels aux fonctions de construction d'arbre, afin de créer la représentation interne du texte.

Examinons l'évolution de la première ligne de l'exemple avec la syntaxe suivante:

```
<en-tête> ::= function <id> <par> : <type> ;
<par>     ::= ( <listepar> ) ;
<listepar> ::= <elem> ; * ;
<élém>    ::= <id> : <type> ;
<type>    ::= real ;
<id>      ::= [a_z]* .
```

Rappelons la suite des tokens en les numérotant:

function	calcul_energie	(	m	:	real	;	c	:	real	)	:	real	;
1	2	3	4	5	6	7	8	9	10	11	12	13	14

La première étape fournit le tableau 3.2.

Tableau 3.2

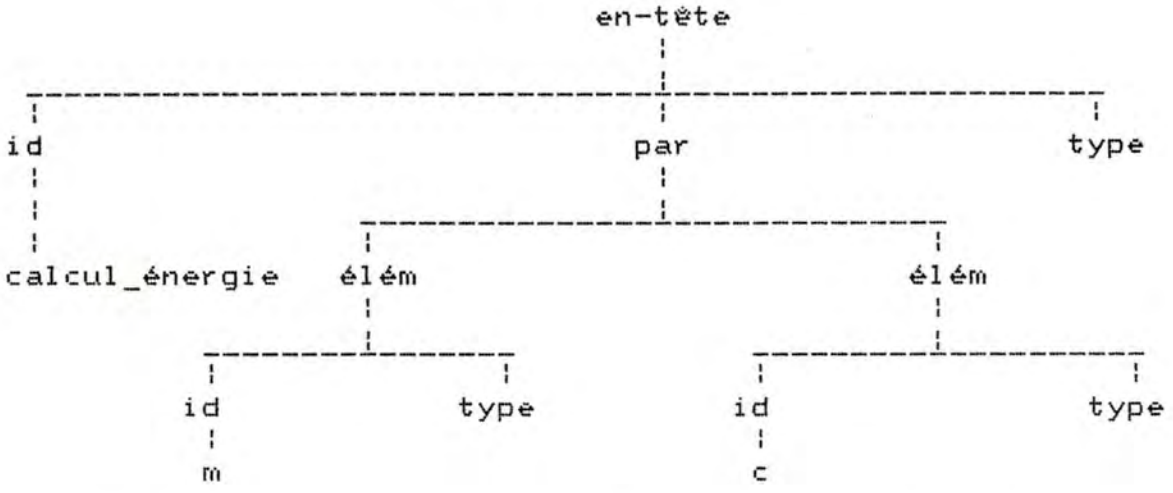
Numéro du token	action:appel pour création d'un sous arbre de type		arguments	résultats
1		----	----	----
2		<id>	"calcul_énergie"	A_1
3		----	----	----
4		<id>	"m"	A_2
5		----	----	----
6	(1)	<type>	----	A_3
	(2)	<élém>	A_2, A_3	A_4
7		----	----	----
8		<id>	"c"	A_5
9		----	----	----
10	(1)	<type>	----	A_6
	(2)	<élém>	A_5, A_6	A_7
11	(1)	<listepar>	A_4, A_7	A_8
	(2)	<par>	A_8	A_9
12		----	----	----
13		<type>	----	A_{10}
14		<en-tête>	A_1, A_9, A_{10}	A_{11}

Il faut remarquer que les mots et symboles réservés ont été filtrés, et qu'ils n'apparaissent plus dans l'arbre.

Un constructeur d'arbres est appliqué aux informations contenues dans le tableau précédant de façon à construire la représentation abstraite. Son rôle est de mettre à la disposition de l'analyseur syntaxique un ensemble de primitives permettant le stockage interne de la représentation.

Exemple: L'application de telles primitives aux informations ci-dessus produit l'arbre du graphe 3.3.

Graphe 3.3



Il faut noter que le texte de départ peut être obtenu en parcourant les feuilles de gauche à droite, et en ajoutant les mots-clé supprimés.

2.5 Parcours de l'arbre

Lorsque la représentation est réalisée, il ne reste plus qu'à parcourir l'arbre pour mémoriser les différents opérateurs et opérandes se trouvant dans les différentes parties contenant les instructions exécutables. Ceci oblige le passage au dessus de toutes les déclarations, elles ne sont pas prises en compte dans les métriques utilisées.

a) Les opérandes:

Les seuls opérandes possibles sont les noeuds génériques, desquels il faut enlever les identificateurs de fonctions et de procédures qui eux sont des opérateurs.

Grâce à cette propriété, il est facile de mémoriser les opérandes puisqu'ils se situent dans certains noeuds génériques.

Exemple concernant la fonction calcul_énergie.

Les opérandes: m , c , 2 et calcul_énergie.

b) Les opérateurs:

Il est plus difficile de noter les opérateurs puisque ceux-ci étant en général des mots réservés, ils sont absents de l'arbre. Il faut donc constituer une table qui contiendra les opérateurs associés à chaque noeud.

Exemple:

Au noeud <par> va être associé l'opérateur "(". Au noeud <affectation> va être associé l'opérateur ":=".

Ainsi, lorsque un noeud est lu, il suffit d'aller consulter la table, pour obtenir les opérateurs associés et les mémoriser.

Exemple concernant la fonction calcul_énergie.

Les opérateurs: := , * , ** , begin .

Cette partie sera développée en détail au chapitre suivant.

3. Conclusions

Cet automate est semblable à l'environnement de programmation MEMTOR[8]. Celui-ci avait été utilisé par A.Schroeder[28] pour réaliser des mesures statiques

et dynamiques de programmes Pascal. Mais aucune mention n'était faite des mesures de complexité. Dans notre cas, cet outil va servir de base à l'automate de calcul des mesures de complexité.

De plus, l'outil[3] utilisé peut réaliser d'autres opérations importantes. Il permet la transformation en arbre abstrait, la visualisation de l'arbre, l'édition syntaxique, l'édition visuelle, la décompilation, ... Toute une série de manipulations de sous-arbres est permise: remplacements, modifications, ... Ceci signifie l'incorporation de primitives de parcours, d'une boîte à outils, de primitives de consultations statiques, de primitives de lecture et d'écriture, ...

La partie utilisée pour réaliser l'automate de mesure est donc très petite, puisqu'elle consiste simplement en la partie transformation, visualisation et décompilation.

CHAPITRE 4:

L'IMPLEMANTATION

L'IMPLEMENTATION

1. La réalisation

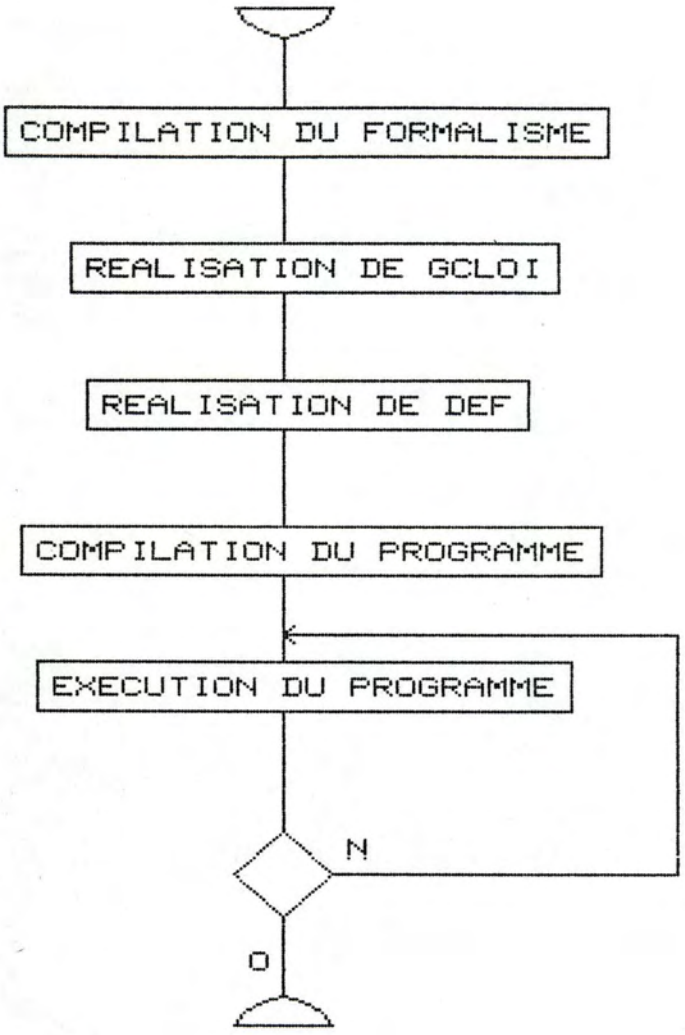
Dans ce chapitre sera présenté la réalisation d'un automate de calcul des métriques de Halstead et McCabe.

Cette réalisation est propre à la syntaxe Pascal. Quelques modifications sont nécessaires si l'on désire passer à une autre syntaxe.

Les étapes de la réalisation sont les suivantes:

- la compilation du formalisme Pascal
- la réalisation du fichier GCLOI
- la réalisation du fichier DEF.C
- la compilation du programme principal (fichier FMEM.C)
- éxécution du programme.

Le séquencement est présenté dans le graphe 4.1.


COMPILATION DU FORMALISME
REALISATION DE GCLOI
REALISATION DE DEF
COMPILATION DU PROGRAMME
EXECUTION DU PROGRAMME
N
O

Graphe 4.1

Examinons chacune des étapes.

2. La compilation du formalisme

Après avoir écrit la syntaxe du formalisme sous forme LDF, celle-ci doit être "transformée". Cette opération fournit un module appelé transformateur.

Pour faciliter l'analyse de la description LDF du formalisme, la première étape sera constituée de la mise en forme arborescente de cette description, suivie de la validation sémantique du programme.

Grâce à ces étapes, le transformateur pourra utiliser les outils syntaxiques pour les textes exprimés dans le formalisme du langage transformé.

Pour arriver à ce résultat, deux types de traitements sont nécessaires: le premier consiste à générer une série d'informations, tandis que le suivant réalise un ensemble de manipulations de codes.

Les informations à générer sont:
- la syntaxe abstraite du formalisme
- les bornes de codification du formalisme et donc le nombre de noeuds de chaque type
- les noms des noeuds du formalisme
- les informations de décompilation
- les spécifications de LEX et YACC pour l'édition visuelle
- les spécifications de LEX et YACC pour la construction interne.

Les manipulations de code à effectuer sont:
- l'utilisation de LEX et YACC avec comme entrées leurs spécifications visuelles et internes
- la compilation des programmes résultats
- l'insertion dans le code de l'éditeur
- le "link" des éditeurs visuels et internes.

Toutes ces opérations sont réalisées par un algorithme réalisé dans le cadre d'un laboratoire de Mr Van Lamsweerde. Il est à noter que toutes les manipulations ne nous sont pas nécessaires. Ce module utilise deux outils offerts par le système d'exploitation UNIX: LEX et YACC. LEX est un analyseur lexical et YACC un analyseur "grammatical".

Pour réaliser cette opération, il faut tout d'abord initialiser un certain nombre d'éléments internes. Ceci est réalisé par l'appel suivant: Jtransfo -i suivi d'un certain nombre de paramètres. Cette première phase réalisée, la syntaxe peut être transformée. Cette deuxième partie est accomplie par Jtransfo NOM et un certain nombre de paramètres où NOM représente le nom du fichier contenant le formalisme. Ces paramètres représentent l'endroit où les renseignements doivent être placés. Celui-ci est dans notre cas le sous-directory /../Transfo/data. Les détails sont présentés dans l'annexe 8.1.

La transformation est relativement lente puisque pour une syntaxe de deux ou trois pages, 30 à 40 minutes de "compilation" sont nécessaires. Elle est de plus très difficile à réaliser de par les exigences de LEX et YACC.

Le formalisme Pascal ainsi que les détails de la réalisation de la transformation sont présentés totalement dans l'annexe 8.1.

3. Réalisation de GCLOI

Cette partie consiste en la construction d'un fichier appelé GCLOI qui va contenir toutes les informations relatives à chacun des noeuds donnés par le formalisme. Ces informations peuvent être représentées sous forme de la table 4.1.

Tableau 4.1

NUMERO DU NOEUD	CODE 1	CODE 2	CODE 3

Le numéro d'un noeud correspond à son numéro de code. Ils sont fournis lors de l'étape précédente dans un fichier appelé NOMABST.TEXT appartenant au sous-directory /../Transfo/data où NOM est le nom du formalisme transformé. Ce fichier reprend aussi les noms des noeuds ainsi que pour chacuns, les noeuds fils avec le numéro de code. Le texte de ce fichier pour la syntaxe Pascal pourra être consulté dans l'annexe 8.2. Le code des noeuds ne correspond pas à l'ordre dans lequel ils surviennent dans le formalisme, en effet, le transformateur regroupe les noeuds de même type. Ainsi, l'ordre de numérotation est le suivant: les noeuds listes, les noeuds quaternaires, les noeuds ternaires, les noeuds binaires, les noeuds unaires et enfin les noeuds zéroaires. Après ces derniers, le transformateur place les noeuds de type générique, annotation, identificateur, formalisme,.... Ceux-ci n'étant pas indiqués dans le fichier NOMABST.TEXT.

Les code 1 et 2 donnent un couple de valeurs indiquant le type de noeud rencontré lors du parcours de l'arbre syntaxique. Plusieurs possibilités sont possibles:

- (0, 0) : le noeud ne contient pas d'opérande ou d'opérateur. Ce noeud est passé lors de la lecture de l'arbre.

- (1, 0) : le noeud n'est pas une liste et contient un seul opérateur.

- (2, 0) : le noeud n'est pas une liste et contient deux opérateurs.

- (3, 0) : le noeud est du type liste.

- (1, 5) : le noeud est une instruction du type "goto ..".

- (0, 1) : le noeud est un identificateur de fonction ou identificateur simple.

La quatrième colonne fait référence à une entrée de la table 4.2. Cette table est introduite dans le programme principal pour permettre à celui-ci de déterminer l'opérateur associé au noeud étudié.

Tableau 4.2

NUMERO	OPERATEUR		NUMERO	OPERATEUR
1	begin		18	not
2	;		19	()
3	:		20	<>
4	:=		21	>
5	if+else		22	<
6	if		23	>=
7	case		24	<=
8	while		25	in
9	repeat		26	+
10	for		27	-
11	to		28	or
12	downto		29	*
13	with		30	/
14	,		31	div
15	[]		32	mod
16	.		33	and
17	^		34	=

Le fichier GCLOI pour la syntaxe Pascal pourra être consulté dans l'annexe 8.3.

4. Réalisation de DEF.C

DEF.C est le nom du fichier contenant les déclarations des codes de certains noeuds importants pour la réalisation du parcours de l'arbre dans le programme principal.
Plutôt que de les introduire dans ce dernier et d'aller y

changer les codes après chaque modification de la syntaxe, il est préférable de les déclarer sous forme de constantes. Le langage C nous permet en plus la possibilité de les déclarer dans un fichier distinct de celui contenant le programme principal. C'est pourquoi ces constantes sont introduites dans ce fichier DEF.C. Il est toutefois nécesaire de recompiler le programme principal après chaque modification du formalisme.

De même que pour l'étape précédente, le fichier NOMABST.TEXT va être très utile pour générer le fichier DEF.C.

Les noeuds utiles sont :

- ceux nécessitant une reconstruction du fait de leur éclatement par le formalisme (réels, string, ...)
- ceux intervenant dans la complexité de McCabe, c'est à dire les noeuds de décision
- le noeud du type goto ..
- celui donnant le début de la partie instruction
- le noeud "instruction d'appel de fonction ou procédure".

Le fichier DEF.C pour la syntaxe Pascal se trouve présenté dans l'annexe 8.4.

5. Compilation du programme principal

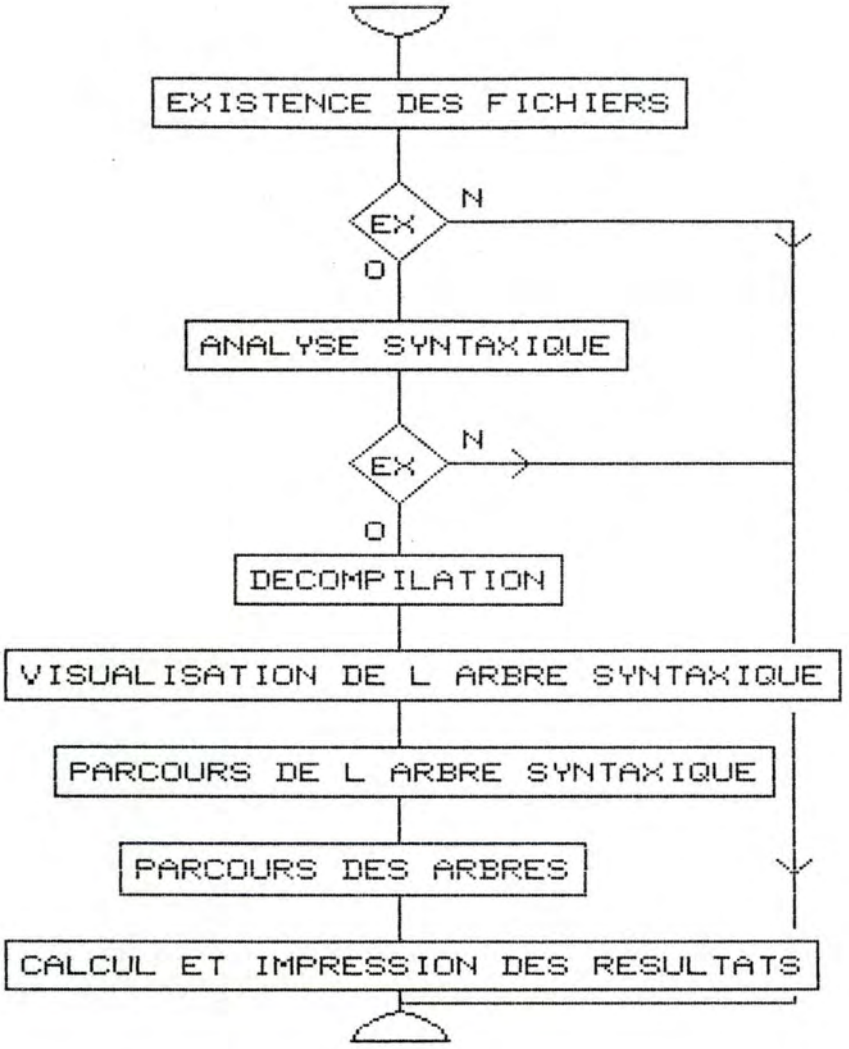
La compilation est réalisée grâce à un fichier de commande appelé MAKEFILE. Celui-ci contient les instructions de compilation, ainsi que les librairies et les options de compilations nécessaires. Le texte de ce fichier de commande se trouve dans l'annexe 8.5.

Le programme principal comprend les étapes suivantes:

- vérification de l'existence des fichiers introduits
- l'analyse syntaxique: transformation en arbre syntaxique

- décompilation de l'arbre syntaxique
- visualisation de l'arbre syntaxique
- parcours de l'arbre syntaxique avec construction des arbres auxiliaires des opérateurs et des opérandes
- parcours des arbres auxiliaires en commençant par celui des opérateurs
- calcul des différentes mesures et impression des résultats.

Le séquencement se fait selon le graphe 4.2.



Graphe 4.2

Examinons en détail chacune des étapes.

5.1 Existence des fichiers

Ce module réalise la vérification de l'existence des fichiers donnés en input lors de l'exécution. Les fichiers sont les suivants: le fichier contenant le programme texte à analyser et ensuite le fichier GCLOI construit plus haut. Lorsque l'un des deux n'existe pas, le programme se termine en indiquant que l'un des fichiers est absent. Par contre lorsque l'existence est vérifiée, le module ouvre les deux fichiers et poursuit.

5.2 L'analyse syntaxique

L'analyse syntaxique consiste comme nous l'avons vu au chapitre précédant, en la transformation du texte du programme de formalisme donné en un arbre, suivant la syntaxe du formalisme étudié (pour nous le pascal) et transformé grâce au transformateur. Le nom d'appel de la fonction est ANSYNT(). Pour que toutes les opérations associées à cet automate se passent bien, il est nécessaire d'introduire dans le texte du programme une référence vers certains directories nécessaires. Ceux-ci ont pour nom Adatdir et Fmsgerr.

Si le texte à analyser possède des erreurs de syntaxe, l'analyseur syntaxique les indique avec le numéro de leur ligne. Le programme principal indique l'absence de référence pour l'arbre syntaxique et se termine. Dans le cas contraire, cette opération renvoie un pointeur vers le sommet de l'arbre syntaxique. Grâce à celui-ci, le parcours de l'arbre est facilité.

Pour que l'analyseur syntaxique reconnaisse le formalisme qu'il doit utiliser, il faut indiquer dans le

fichier texte du programme à analyser suivant quelle syntaxe il a été écrit.Si le formalisme s'appelle GC, il faudra donc indiquer comme suit:

```
(* GC *)
program test;
......
```

En effet, le transformateur mémorise tous les formalismes utilisés. Les appels à chacune de ces syntaxes sont intégrés dans le programme exécutable par l'analyseur syntaxique. Cette mémorisation implique qu'à tout moment, tous les formalismes transformés peuvent être utilisés.

Il faut noter que l'analyseur ne travaille qu'avec les deux premiers caractères des noms de formalismes. L'équivalence entre le nom entier du formalisme et la variable réduite est donnée dans un fichier dont le nom est NAMES se trouvant dans la directory /../Transfo/data.

5.3 Décompilation

Le décompilateur fournit à partir de l'arbre syntaxique la version texte du programme analysé avec indentation, passage à la ligne, saut de lignes,...

Pour réaliser clairement cette décompilation, c'est à dire pour que le texte obtenu soit clair, il faut introduire dans la syntaxe avant transformation certains symboles qui indiqueront les opérations à effectuer.

Ces symboles sont:

- $: le décompilateur doit passer à la ligne avant (après) impression du séparateur

- # : le décompilateur doit passer à la ligne et indenter avant (après) impression du séparateur. L'indentation reviendra à son état original après la décompilation du

noeud suivant le symbole

- @ : le décompilateur doit passer i lignes avant (après) l'impression du séparateur

- ^ : le décompilateur ne doit pas insérer de blanc avant (après) l'impression du séparateur.

Cette décompilation est réalisée dans le programme principal grâce à la fonction Fdufil ayant comme paramètre le pointeur vers le sommet de l'arbre syntaxique. Cette opération consiste simplement en un parcours de l'arbre syntaxique, avec ajout des mots-clés et impression.

5.4 Visualisation de l'arbre

Ce module consiste en la visualisation de l'arbre syntaxique. La représentation est la suivante:

- chaque noeud est représenté sur une ligne avec son nom et son numéro de code ainsi que sa longeur et sa valeur pour les génériques.

- ses noeuds fils sont indentés par rapport à lui de deux caractères vers la droite.

- les noeuds frères sont situés juste en dessous du noeud frère précédent.

Des exemples de cette visualisation peuvent être consultés en annexe 8.7.

Cette visualisation permet de vérifier comment le programme à analyser a été décomposé et si les règles de syntaxe du formalisme sont valables.

5.5 Parcours de l'arbre syntaxique

Dans ce module, un parcours de l'arbre syntaxique est réalisé dans le but d'y détecter les opérateurs et les opérandes. Ceux-ci sont placés dans deux arbres distincts permettant une recherche plus facile et un classement alphabétique en sortie. Chaque noeud possède deux composantes: une première servant à la mémorisation de la valeur et une seconde mémorisant le nombre de fois que celle-ci a été rencontrée depuis le début du parcours.

Ce parcours est réalisé par une procédure nommée POSITION récursive ayant comme paramètre un pointeur vers le noeud de l'arbre à analyser.

a) L'algorithme POSITION

Comme les déclarations ne sont pas prises en compte pour le calcul de la complexité, il faut donc aller se positionner sur chaque début de "partie instruction". Lorsque ce noeud est trouvé, on peut passer à l'analyse plus détaillée de l'arbre. Cette étude est réalisée par une nouvelle procédure récursive LIREFILS ayant un pointeur vers le noeud à traiter comme paramètre.

L'algorithme POSITION est le suivant:

```
1. si le code du noeud est distinct du code de début
   d'instruction
   alors  si le noeud n'est pas un générique
          alors passage au noeud fils
                retour en 1;
          passage au noeud frère;
          retour en 1;
   sinon on passe à LIREFILS .
```

b) L'algorithme LIREFILS

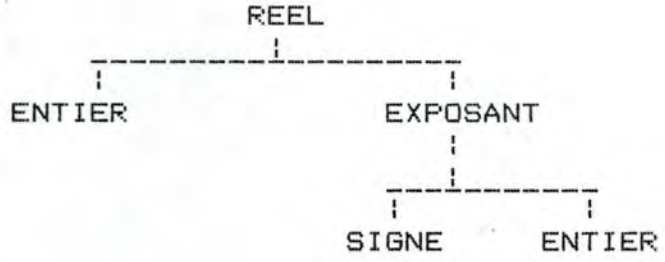
Les noeuds font donc partie du sous-arbre dont le sommet est le noeud "partie instruction". Ils sont examinés l'un après l'autre.

La première opération effectuée consiste suivant le noeud à aller rechercher dans le fichier GCLOI les trois nombres dont nous avons déjà parlé. Le code du noeud sert de clé d'accès, en effet, la recherche se fait à l'aide d'un SEEK.

Suivant les deux premiers codes obtenus, on peut facilement dire si des opérandes ou des opérateurs interviennent dans ce noeud.

Si le noeud est un opérande, le nombre d'opérandes est augmenté d'une unité et sa valeur est introduite dans l'arbre correspondant avec un poids de un. Si cette valeur est déja présente , le poids du noeud correspondant est augmenté de un. Il se peut que l'opérande soit décomposé par l'analyseur syntaxique, il faut alors à l'aide de la procédure CALCUL la recomposer avant de l'introduire dans l'arbre correspondant. En Pascal, ce sera le cas pour les nombres réels, les strings, les nombres entiers signés. Prenons par exemple un réel de la forme 4e-5. Ce nombre est composé suivant la syntaxe de trois parties: 4, e et -5, cette dernière étant elle-même décomposée en deux parties: - et 5. A ce sous-arbre correspond donc le grphe 4.3.

Graphe 4.3



La procédure CALCUL consiste à calculer les valeurs des différentes feuilles du sous-arbre et à ajouter le symbole de l'exposant.

Si le noeud fait intervenir un opérateur, on recherche la valeur de celui-ci grâce au troisième code obtenu de GCLOI. Deux cas sont possibles. Si le noeud n'est pas une liste, le nombre d'opérateurs est alors augmenté d'une unité et la valeur est introduite dans un deuxième arbre

avec un poids un .Si celle-ci est déjà présente, la deuxième composante du noeud est augmentée de un. Par contre si le noeud est une liste, le nombre de fois que l'opérateur intervient égale le nombre de noeuds fils moins un. Supposons que cette valeur soit n, on doit alors augmenter le nombre total d'opérateurs de n, introduire la valeur avec un poids n si elle n'est pas déjà présente sinon augmenter la deuxième composante du noeud de n.

Si le noeud comprend deux opérateurs, la procédure est la même avec chacun de ceux-ci.

Pendant le parcours de l'arbre, on comptabilise le nombre de passages dans des noeuds de décision. Deux compteurs sont ainsi utilisés, un pour le calcul du nombre de McCabe et l'autre pour le calcul de Cycmin. La différence entre les deux, résidant dans la manière de compter les alternatives du CASE. Dans le premier cas, on fait intervenir le nombre d'alternatives moins une. Par contre, on ne comptera qu'une alternative dans le second cas.

L'algorithme est finalement le suivant:

```
1. recherche des trois codes dans GCLOI
2. si le noeud est un opérande
   alors . le nombre total est augmenté de un
         . insertion de la valeur dans l'arbre des opérandes
           avec un poids un
   si le noeud fait intervenir un opérateur
   alors si le noeud n'est pas une liste
         alors .  le nombre d'opérateurs est augmenté de un
                . la valeur est introduite dans l'arbre
         si le noeud est une liste avec n+1 fils
         alors .  le nombre total égale le nombre total plus n
                . la valeur est introduite dans l'arbre avec
                  un poids n
   si le noeud fait intervenir deux opérateurs
   alors .  le nombre d'opérateurs est augmenté de deux
         .  les deux valeurs sont introduites dans l'arbre
            avec un poids un
3. si le noeud est un générique
   alors on passe au noeud frère
         retour en 1
4. on parcourt le noeud fils
   retour en un.
```

Il faut encore signaler qu'après modification du formalisme, il est nécessaire de modifier GCLOI et DEF.C et recompiler le programme principal. Si cela n'est pas réalisé, des problèmes vont survenir lors de la réalisation de ce module.

5.6 Parcours des arbres auxiliaires

Le parcours des deux arbres auxiliaires est fait de façon identique. Comme il s'agit d'arbres binaires, il suffit de parcourir les sous-arbres de gauche, les sommets et ensuite les sous-arbres de droite. Lorsque un noeud est parcouru, un compteur est augmenté d'une unité. La valeur finale de celui-ci donne le nombre d'opérateurs distincts respectivement d'opérandes distincts. Une liste de ces opérateurs respectivement de ces opérandes est réalisée en même temps que le comptage. Ces deux nombres sont importants pour le calcul de la complexité de Halstead.

5.7 Calcul des complexités

Ce module calcule la complexité selon McCabe, l'intervalle Cycmin-N1, la longueur, le volume, le niveau et la complexité du programme selon Halstead.

La première mesure est obtenue facilement à partir du premier compteur utilisé lors du parcours de l'arbre syntaxique puisqu'il suffit de lui ajouter un.

La deuxième est composée du deuxième compteur utilisé lors du parcours de l'arbre syntaxique augmenté de un et du nombre total d'opérateurs.

Les dernières sont obtenues par application des formules proposées par Halstead, présentées dans le

chapitre 2.

Le texte du programme principal est présenté dans l'annexe 8.6.

6 Exécution du programme principal

L'exécution se fait simplement en indiquant la commande suivante: FMEM TEST GCLOI où TEST est le nom du fichier contenant le programme à analyser. Celle-ci permet une impression des résultats dans un fichier dont le nom est IMPR grâce au système d'exploitation UNIX. La commande est alors la suivante: FMEM TEST GCLOI > IMPR.

7 Conclusion

Lorsque l'on dispose de l'analyseur syntaxique, l'implémentation de l'outil est très facile.

Puisque l'arbre syntaxique est déjà fourni, il ne suffit plus que de le parcourir pour compter les opérandes et les opérateurs. La suite consiste en l'application de simples formules.

Il faut encore une fois noter qu'une modification de la syntaxe entraine une retransformation de cette dernière, suivie par la modification de GCLOI, DEF.C et par la compilation de FMEM.C grâce à MAKEFILE.

CHAPITRE 5:

ETUDE D'UN CAS

ETUDE D'UN CAS

1. Présentation du problème

Après une étude théorique de la complexité et après la construction d'une méthode de calcul des différentes métriques, il apparait intéressant de l'appliquer sur un exemple.

Le problème choisi est relativement connu, puisqu'il consiste à placer huit reines sur un échiquier en respectant les règles de l'échec: aucune des reines ne peut prendre une des autres.

Deux textes différents ont été introduit dans l'"outil de mesure" décrit au chapitre précédent. La première de ces versions a été construite en deuxième candidature en science mathématique lors de la première année de programmation en Pascal. La deuxième version a été réalisée par le même programmeur quelques années plus tard.

On pourra donc constater si les années de pratique du Pascal ont une influence sur les différentes mesures réalisées. Les causes des différences entre les deux algorithmes seront étudiées plus en détail dans un dernier paragraphe.

2. Les huit reines

Les résultats sont calculés en introduisant les deux programmes dans l'automate de calcul. Ils sont très

facilement obtenus après un temps de calcul relativement important. En effet, l'"outil" doit construire pour chacun des deux algorithmes l'arbre syntaxique. Celui-ci est important puisqu'il est reproduit sur 11 à 12 feuilles de listing.

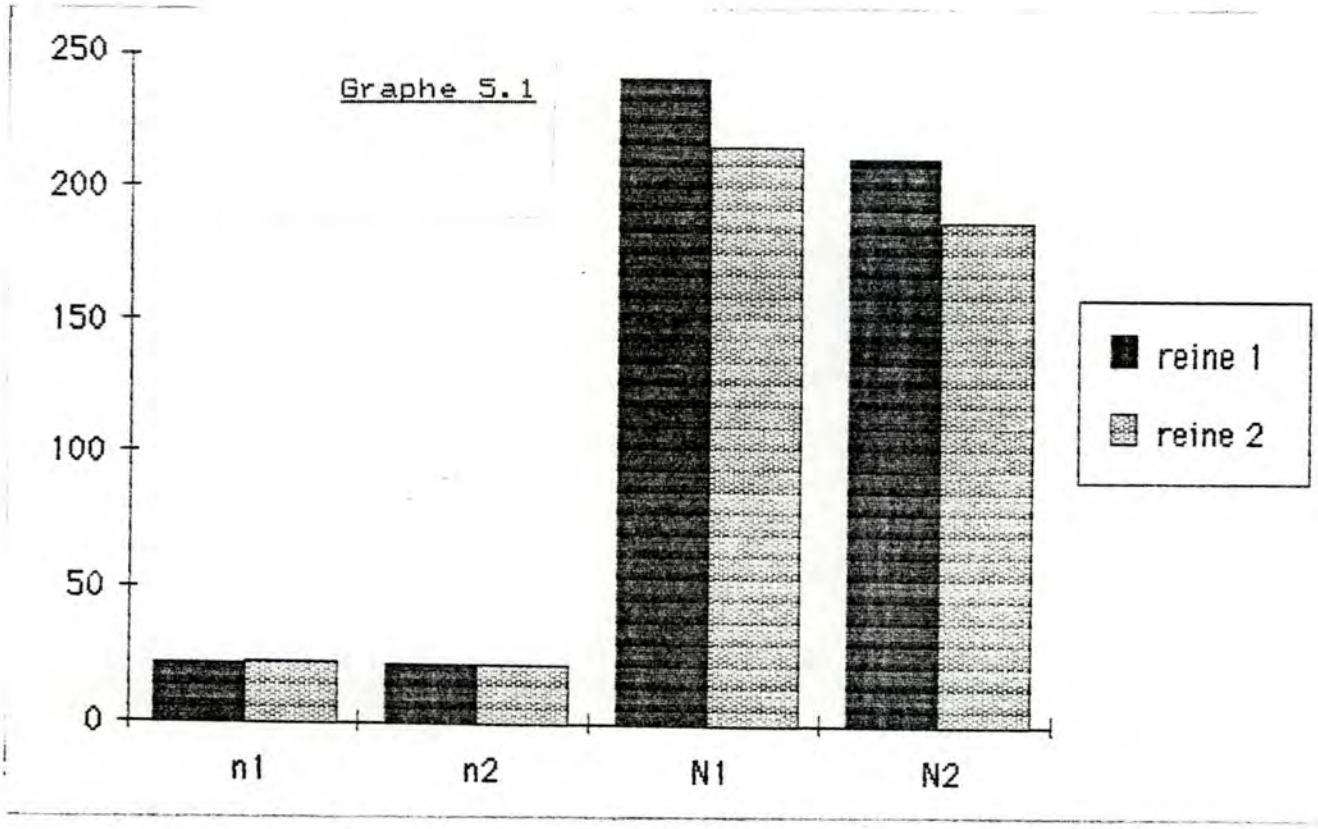
Les résultats des algorithmes sont présentés dans le tableau 5.1:

Tableau 5.1

	n1	n2	N1	N2	N	V	L	E	v(G)
reine 1	22	22	241	212	196.22	2473.12	0.009	262151	22
reine 2	23	22	216	188	202.15	2218.71	0.01	218038.6	21

On remarquera les différences qu'il peut exister entre la première et la deuxième version. Celles-ci sont présentées pour n_1, n_2, N_1 et N_2 dans le graphe 5.1.

Graphe 5.1



Malgré l'introduction d'un opérateur, les nombres d'opérateurs et d'opérandes diminuent de plus ou moins 10 pour cent. Cela se traduit par une augmentation de la longueur N correspondant à la formule (1.2), cette formule n'emploie que n_1 et n_2. Par contre, le volume et la complexité diminuent de façon sensible. Cela est du à la forte diminution de N_1 et N_2. Mais comme les deux programmes réalisent la même chose, il est normal que les niveaux de programme soient identiques.

Les années de pratique supplémentaire ont donc porté leurs fruits. Cela confirme les hypothèses faites par les différents chercheurs dont nous avons déjà parlé au chapitre 2.

Mais d'où proviennent ces différences?

3. Les causes

Avant d'examiner les textes des deux programmes, établissons les formules qui à partir des mesures réalisées sur une version permettent de passer à de nouvelles formules correspondant à une nouvelle version du même programme sans passer par l'analyseur syntaxique. Ces formules sont générales. Elles peuvent s'appliquer à toutes les transformations de programmes.

3.1 Les formules de transformation

Une première version du programme est introduite dans l'analyseur syntaxique. Elle produit les variables suivantes: n_1, n_2, N_1 , N_2 les quatres nombres de Halstead, N une approximation de la longueur, V le volume, L une estimation du niveau, E la complexité du programme .

Soient n_1',n_2',N_1',N_2',N',V',L' et E' variables correspondantes associées à la deuxième version du programme. En observant celui-ci,il est assez facile de déterminer les variables suivantes:

- d_1 = le nombre de nouveaux opérateurs introduits
- d_2 = le nombre de nouveaux opérandes introduits
- D_1 = le nombre de nouvelles occurences d'opérateurs introduites
- D_2 = le nombre de nouvelles occurences d'opérandes introduites.

Les quatre nombres de base de la seconde version s'obtiennent facilement en fonction des mesures de

la première version en utilisant les quatre définitions précédantes. On obtient:

$$\begin{aligned} n_1' &= n_1 + d_1 \\ n_2' &= n_2 + d_2 \\ N_1' &= N_1 + D_1 \\ N_2' &= N_2 + D_2. \end{aligned} \qquad (5.1)$$

En utilisant la formule (1.2), (1.3), (1.7) et (5.1), on peut déterminer facilement les mesures suivantes:

- une estimation de la longueur:

$$N' = (n_1+d_1)\log_2(n_1+d_1) + (n_2+d_2)\log_2(n_2+d_2) \qquad (5.2)$$

- le volume:

$$V' = (N_1+N_2+D_1+D_2)\log_2(n_1+n_2+d_1+d_2) \qquad (5.3)$$

- une estimation du niveau du programme:

$$L = \frac{2\,(n_2+d_2)}{(n_1+d_1)\,(N_2+D_2)} \qquad (5.4)$$

3.2 Examen des programmes des reines

Examinons les deux textes. Les différences suivantes sont observées:

- une nouvelle procédure a été introduite remplaçant deux parties identiques

- trois boucles "while" sont transformées en boucles "for"

- trois "if then else" sont remplacés par des "if then".

Ces trois points vont être examinés séparément et donner lieu à des formules de transformations. Ces dernières étant établies grâce aux formules (5.1), (5.2), (5.3) et (5.4), elles sont applicables dans tous les cas semblables et non seulement dans le cas du programme des huit reines..

La transformation du "while"

Deux petits programmes vont servir à déterminer les différentes mesures. La première version contiendra une boucle "while", l'autre sera constituée par une boucle "for".

Le premier programme contient la partie suivante:

```
i:=1;
while i<=n do begin
     def:=def+1;
     i:=i+1
end.
```

Le deuxième est simplement constitué par la boucle suivante:

```
for i:=1 to n do def:=def+i.
```

On peut observer que le premier programme nécessite une initialisation de la variable de boucle (i) et une incrémentation (i:=i+1) de cette dernière à l'intérieur même de la boucle.

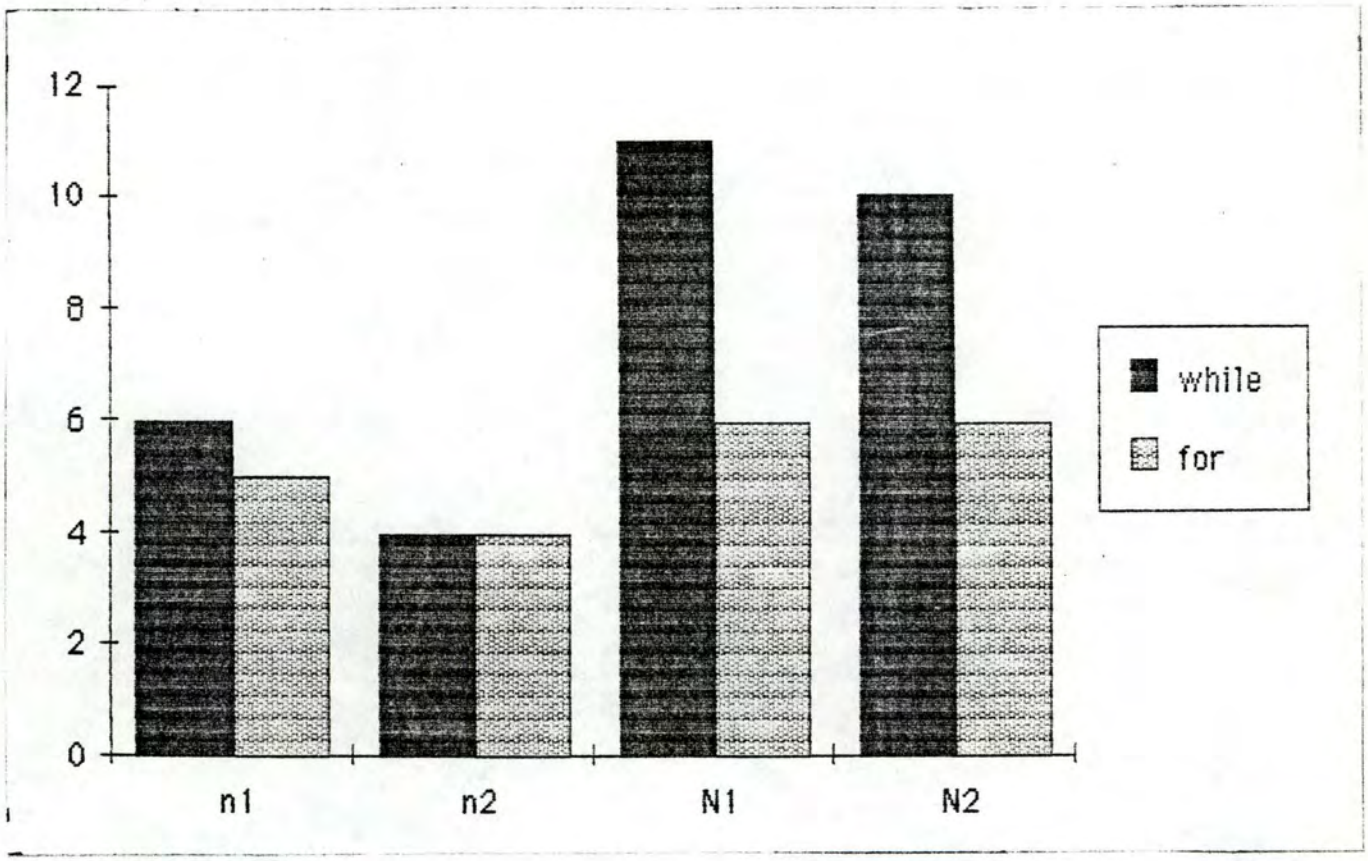
Regroupons les résultats obtenus dans le tableau 5.2 pour faciliter l'observation des différences.

Tableau 5.2

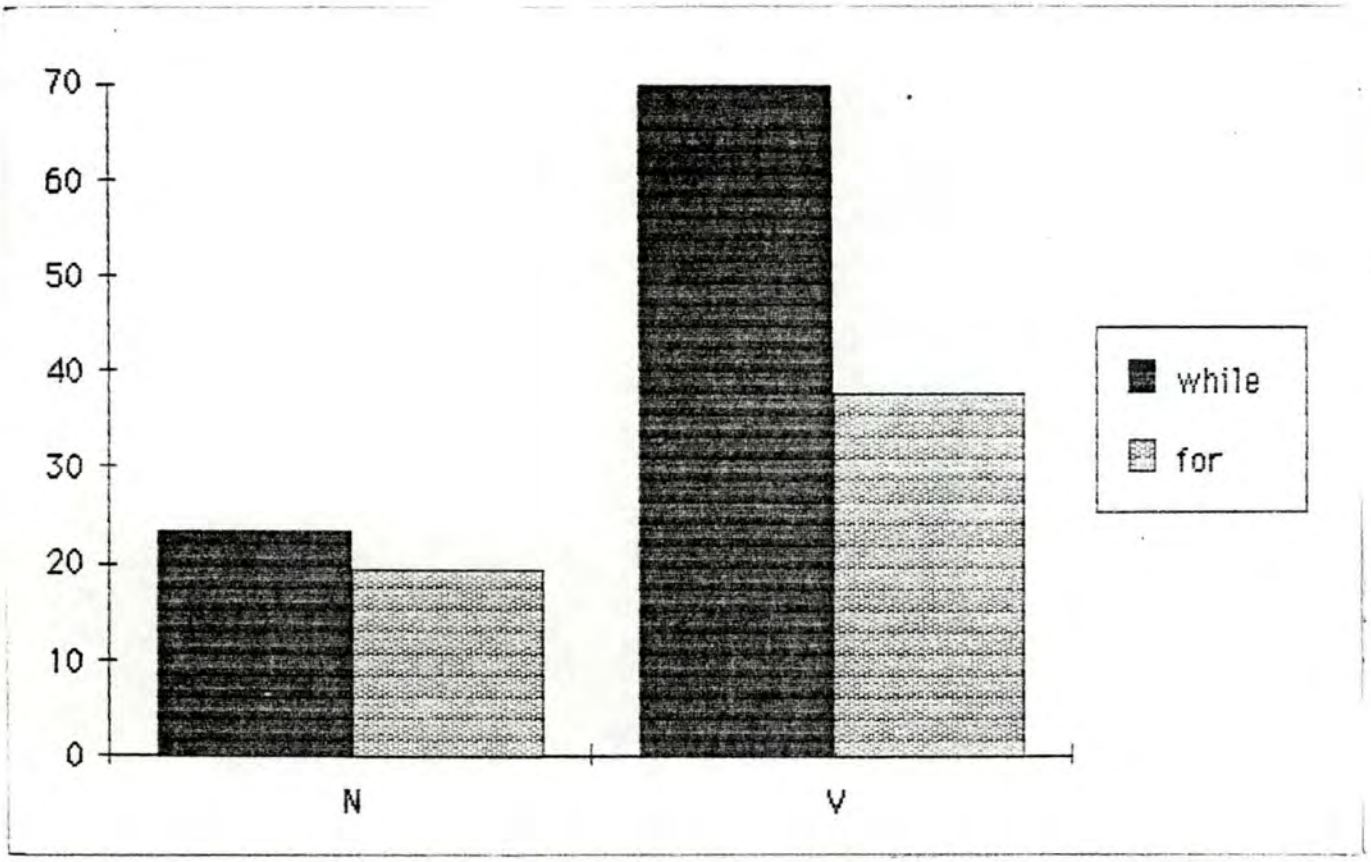
	n1	n2	N1	N2	N	V	L	E	v(G)
while	6	4	11	10	23.5	69.8	0.13	523.2	2
for	5	4	6	6	19.6	38	0.27	142.65	2

On observe une diminution très importante de la complexité. Le rapport entre E' et E est de 1/5. Cette diminution provient de la chute des nombres d'opérateurs et d'opérandes (graphe 5.2). En effet, E est proportionnel à V et inversémment proportionnel à L. Comme V' est inférieur à V et L' est supérieur à L, E' devient plus petit que E. Ces variations sont visibles pour la longueur et le volume dans le graphe 5.3. La différence de grandeur entre V et L rend impossible la visualisation correcte de leur variation sur un même graphe.

Graphe 5.2



Graphe 5.3



Il est également important d'observer les formules de transformations afin de pouvoir les utiliser et de ne plus passer par l'analyseur syntaxique.

Les différences peuvent être calculées:

$$d_1 = -1$$
$$d_2 = 0$$
$$D_1 = -5$$
$$D_2 = -4 \ .$$ (5.5)

Ceci permet de donner les formules de V' et L' à partir des coéfficients du premier programme:

- le volume:

$$V' = (N_1+N_2-9)\ \log_2(n_1+n_2-1) \qquad (5.6)$$

- le niveau du programme:

$$L' = \frac{2\ n_2}{(n_1-1)\ (N_2-4)} \qquad (5.7)$$

A l'aide de ces deux formules, il est très facile de calculer E' comme le rapport de V' sur L'.

Transformation du "if"

Cette partie va se dérouler de la même façon que la transformation précédente. Deux programmes vont être introduits dans l'analyseur syntaxique. Les deux programmes sont les suivants:

- le premier:

```
if i<n
then begin
  def:=def+1;
  i:=i+1
end
else i:=i+1;
```

- le second:

```
if i<n then def:=def+1;
i:=i+1;
```

On voit facilement que le deuxième programme est préférable, puisque dans le premier, on retrouve une même instruction d'incrémentation (i:=i+1) dans les deux parties du "if".

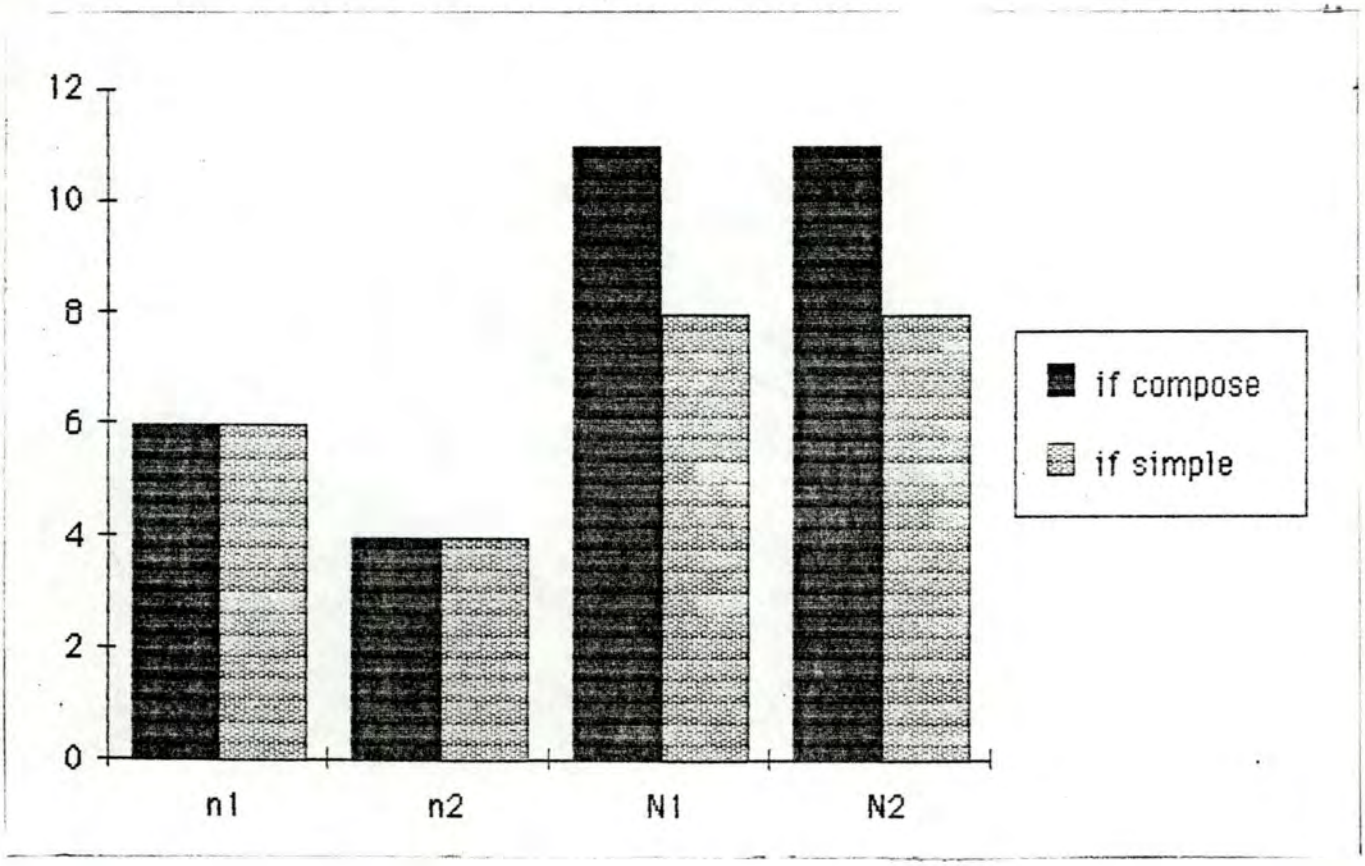
Regroupons les résultats dans le tableau 5.3.

Tableau 5.3

	n1	n2	N1	N2	N	V	L	E	v(G)
if compose	6	4	11	11	23.51	73.08	0.12	602.93	2
if simple	6	4	8	8	23.51	53.15	0.17	318.91	2

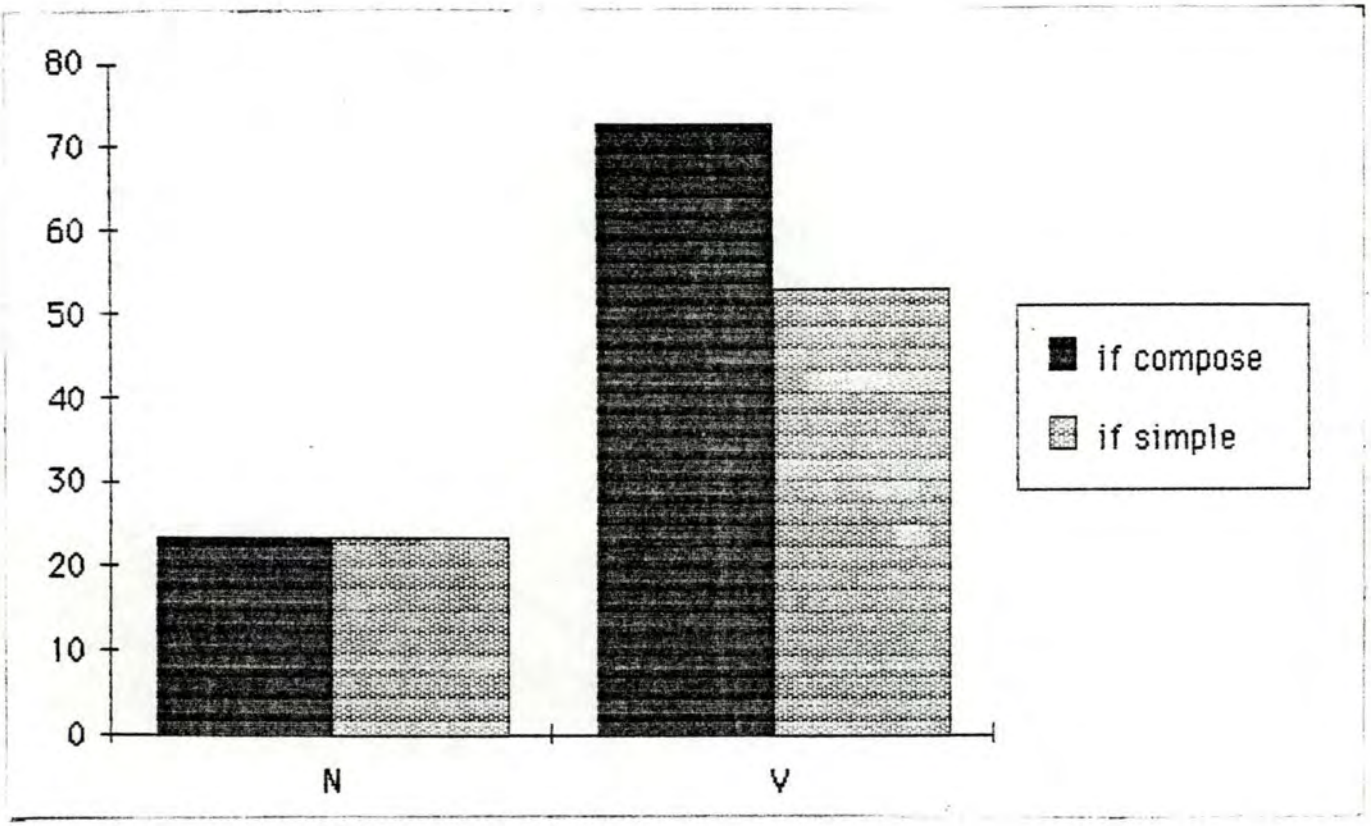
Les nombres d'opérateurs et d'opérandes ont diminué, comme le graphe 5.4 le montre.

Graphe 5.4



Ceci entraine une diminution du volume et une augmentation du niveau du programme, et a donc de nouveau pour effet, une diminution de la complexité E' . Le rapport entre E' et E est de 1/2. Cette différence est très importante et peut amener des coéfficients de complexité beaucoup plus grands dans les programmes où cette transformation n'a pas été réalisée. Il ne faut pas oublier que E est synonyme d'erreurs, d'où la nécessité de diminuer au maximum la complexité. Le graphe 5.5 montre les différences entre N et N' et entre V et V'. Les grandeurs de V et L étant très différentes, il est de nouveau impossible de les associer dans un même graphe.

Graphe 5.5



La deuxième étape consiste en l'observation des formules de transformations. Commençons par donner les quatre nombres de Halstead.

$$n_1' = n_1$$
$$n_2' = n_2$$
$$N_1' = N_1 - 3$$
$$N_2' = n_2 - 3 \ . \qquad (5.8)$$

Poursuivons avec V' et L'.

- le volume:

$$V' = (N_1+N_2-6)\ \log_2(n_1+n_2) \qquad (5.9)$$

- le niveau du programme:

$$L' = \frac{2\ n_2}{n_1\ (N_2-3)} \qquad (5.10)$$

Ces formules permettent de calculer E'.

Remplacement par procédures

Dans cette partie, le remplacement d'instructions par une procédure va être étudié en détail. Les tests vont s'établir de la façon suivante:

remplacement de :

- une série d'instructions par une procédure
- deux séries identiques par deux appels de procédeure
- trois séries
- quatre séries
- et enfin, cinq séries.

Cela va permettre d'étudier l'évolution de la complexité dans les différents cas.

Regroupons les résultats obtenus dans le tableau 5.4.

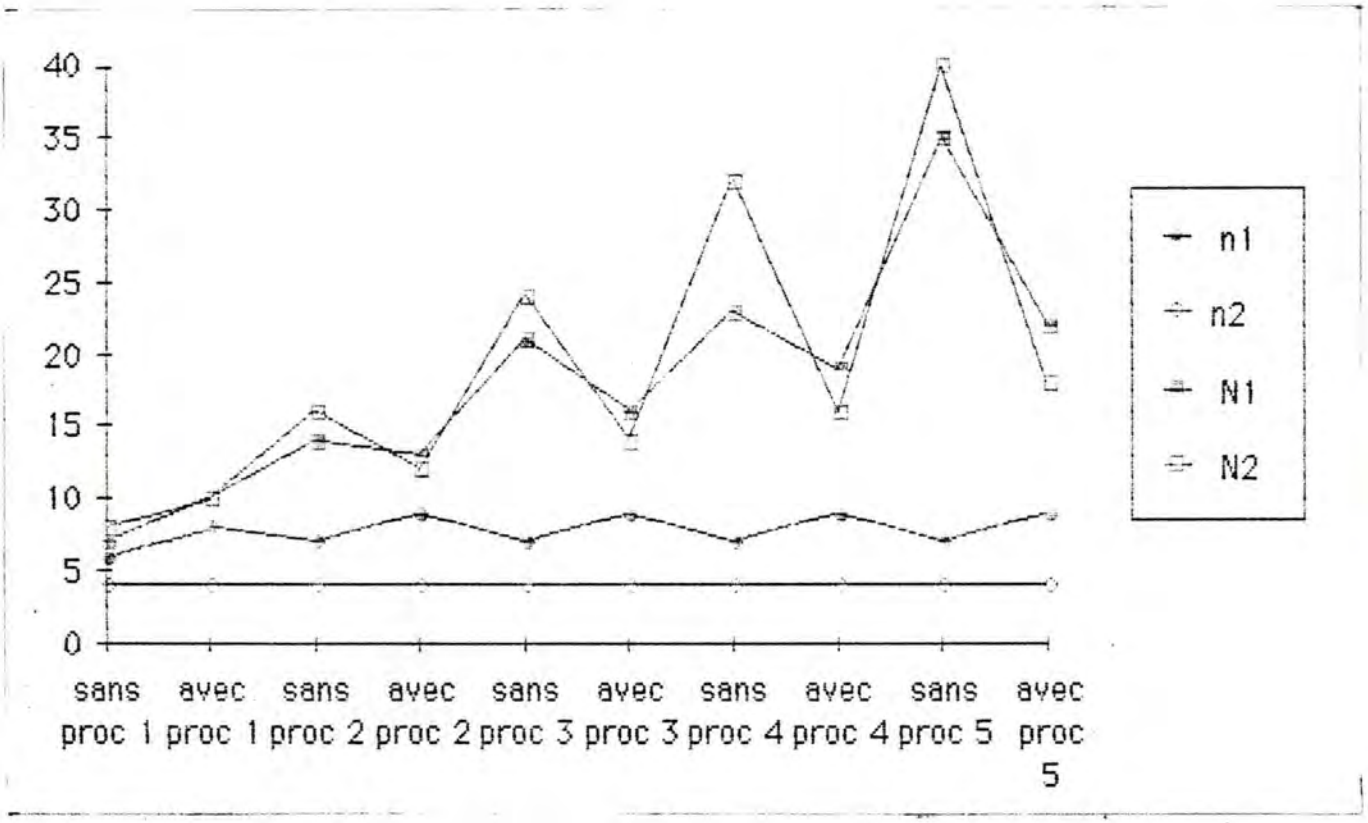
Tableau 5.4

	n1	n2	N1	N2	N	V	L	E	v(G)
sans proc 1	6	4	7	8	23.5	49.83	0.17	298.97	2
avec proc 1	8	4	10	10	32	71.7	0.1	716.99	2
sans proc 2	7	4	14	16	27.7	103.8	0.07	1453	3
avec proc 2	9	4	13	12	36.5	92.51	0.07	1248.9	2
sans proc 3	7	4	21	24	27.7	155.7	0.05	3269.2	4
avec proc 3	9	4	16	14	36.5	111	0.06	1748.5	2
sans proc 4	7	4	23	32	27.7	207.6	0.03	5811.9	5
avec proc 4	9	4	19	16	36.5	129.5	0.06	2331.3	2
sans proc 5	7	4	35	40	27.7	259.5	0.03	9081	6
avec proc 5	9	4	22	18	36.5	148	0.05	2997.4	2

Le mot "sans" signifie qu'aucun appel de procédure n'a été utilisé, par contre le mot "avec" signifie que les instructions ont été remplacées par des appels.

Les différences peuvent être observées dans le graphe 5.6.

Graphe 5.6

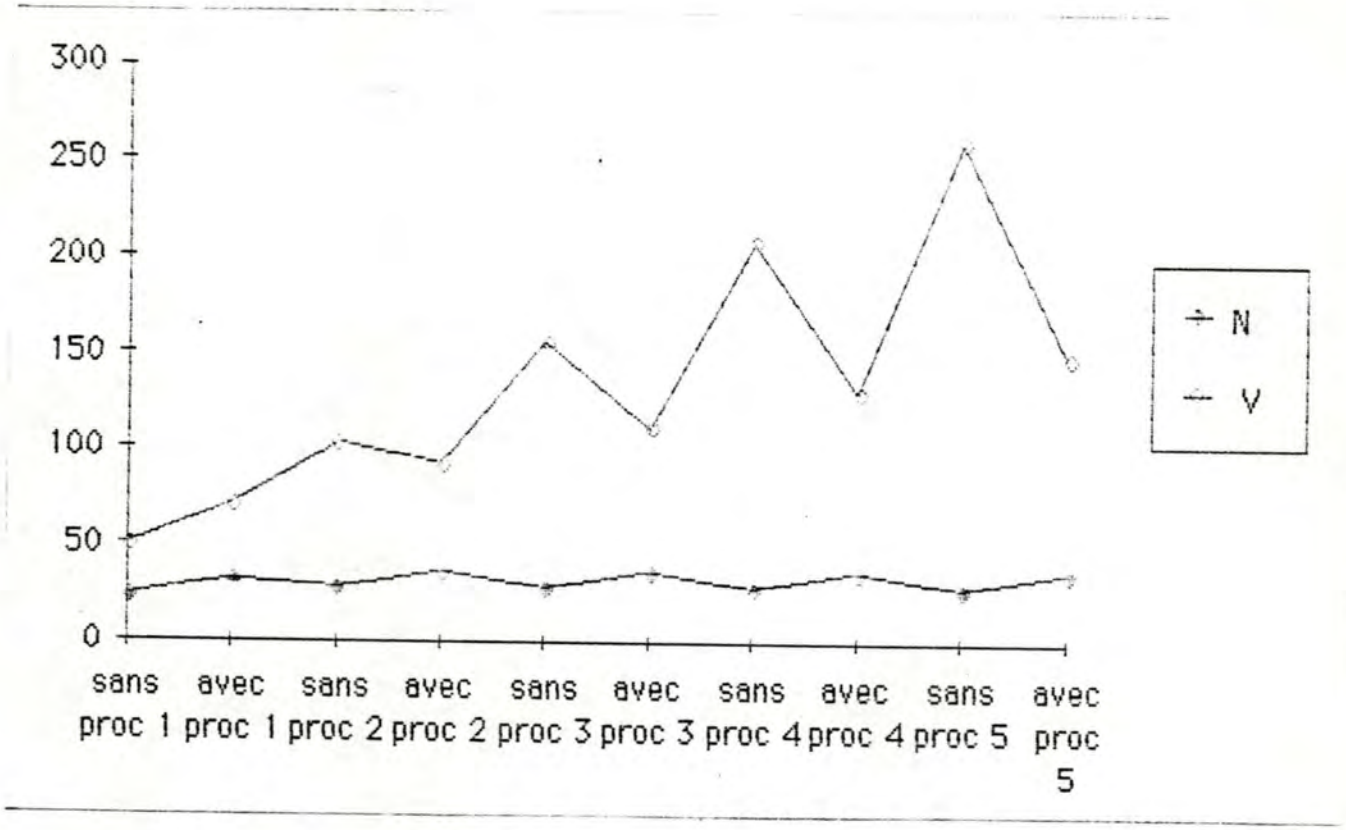


L'évaluation des graphiques se fait par l'observation de deux abscisses groupant un programme sans remplacement et un programme avec procédures.

Il faut observer que le remplacement d'une seule série apporte une augmentation des quatre nombres de Halstead, ce qui se traduit par une augmentation de complexité. Il n'est donc pas intéressant dans ce cas de remplacer une série d'instructions par la constitution d'une procédure et d'un appel. Par contre au dessus de deux appels, cela devient rentable puisque l'on observe dans chacun des cas une diminution de la complexité. On peut même

aller plus loin, plus le nombre de remplacements est grand plus il est indispensable d'utiliser une procédure . Le rapport de E' sur E évolue en effet de 3/1 à 1/4. Comme dans les cas précédants, V et L ne peuvent être regroupés dans un même graphe. Le graphe 5.7 montre les variations de N et V.

Graphe 5.7



Les formules de transformations sont plus difficiles à donner, elles dépendent du nombre de remplacements, du nombre de séparateurs (,) dans l'appel, du nombre d'opérandes et du nombre d'opérateurs dans le corps de la procédure. Posons donc:

- R le nombre de remplacements
- S le nombre de séparateurs dans l'instruction d'appel
- p_1 le nombre d'opérateurs dans le corps de la procédure
- p_2 le nombre d'opérandes dans le corps de la procédure.

On obtient donc pour les quatre nombres principaux:

$$\begin{aligned} n_1' &= n_1 + 1 \text{ si la virgule est utilisée ailleurs} \\ &\quad\; n_1 + 2 \text{ sinon} \\ n_2' &= n_2 \\ N_1' &= N_1 + 1 + R\,(S + 1) - (R - 1)\,p_1 \\ N_2' &= N_2 + R\,(S + 1) - (R - 1)\,p_2. \end{aligned} \qquad (5.11)$$

Un nouvel opérateur (le nom de la procédure) est ajouté, ainsi que la virgule si celle-ci n'est à aucun moment présente dans le texte du programme. Le nombre d'opérandes distincts ne varie pas puisque tous ceux qui sont utilisés dans le corps de la procédure sont aussi présents dans le programme sans appels. Par contre, N_1 est augmenté d'une unité correspondant au "begin" de la procédure. Il faut encore ajouter les appels: R ajout de (S + 1) opérateurs où S correspond au nombre de séparateurs et le 1 correspond au nom d'appel de la procédure. Il ne faut pas oublier d'enlever les parties remplacées par un appel, c'est à dire (R - 1) remplacement de p_1 opérateurs, puisqu'il subsiste les opérateurs du corps de la procédure. Pour le calcul du nombre total d'opérandes, il est nécessaire d'ajouter R appels contenant (S + 1) opérandes et d'enlever les R parties contenant p_2 opérateurs remplacées par les appels. Il ne faut toutefois pas oublier de compter p_2 opérandes du corps de la procédure.

Ayant ces formules, il est facile de calculer les différentes autres mesures.

Association des trois types de modifications

Soit un plus gros programme dont on connait les différents nombres caractéristiques. Supposons que l'on puisse réaliser sur ce programme h_w transformations du type "while", h_i transformations du type "if" et R

remplacements d'une partie commune contenant p_1 opérateurs et p_2 opérandes par des appels de procédures avec S séparateurs. On peut facilement avoir une idée des quatre nombres principaux. Attention, il peut cependant exister une différence due à un point virgule non comptabilisé. Les formules de transformation sont les suivantes:

$$\begin{aligned} n_1' &\triangleq n_1 \\ n_2' &\triangleq n_2 \\ N_1' &\triangleq N_1 + 1 + R\,(S+1) - (R-1)\,p_1 - 5\,h_w - 3\,h_1 \\ N_2' &\triangleq N_2 + R\,(S+1) - (R-1)\,p_2 - 4\,h_w - 3h_1. \end{aligned} \quad (5.11)$$

Appliquons ces résultats aux programmes des reines:

$$\begin{aligned} h_w &= 3 \\ h_1 &= 3 \\ R &= 2 \\ S &= 3 \\ p_1 &= 11 \\ P_2 &= 11 \end{aligned} \quad (5.12)$$

On obtient donc

$$\begin{aligned} n_1' &\triangleq 22 \\ n_2' &\triangleq 22 \\ N_1' &\triangleq 241 + 1 + 2\,(3+1) - (2-1)\,11 - 15 - 9 = 215 \\ N_2' &\triangleq 212 + 2\,(3+1) - (2-1)\,11 - 12 - 9 = 188. \end{aligned} \quad (5.13)$$

On observe la petite différence déjà signalée plus haut.

On peut obtenir facilement une approximation des quatre nombres à partir des mesures réalisées sur le programme non encore modifié. Il suffit de compter le nombre de transformations de ce type et d'appliquer la formule (5.11). Il est alors très facile de calculer la complexité. Ceci permet à l'utilisateur d'avoir une approximation de l'effort de programmation, du nombre d'erreurs, du temps de programmation, ...

4. Conclusion

Pour deux programmes réalisant la même fonction mais composés de légéres différences. Il est possible de calculer à partir d'une des versions la complexité de la seconde sans passer par l'analyseur syntaxique. Les formules de passage permettent ce calcul.

Il faut cependant insister sur la différence qu'il peut exister entre l'estimation et la mesure de la complexité.

CHAPITRE 6:

DES APPLICATIONS

1. Introduction

Actuellement, beaucoup de temps est dépensé pour la maintenance de software existant plutôt que pour la création de nouveaux programmes. En réalité, les ressources investies durant les phases de maintenance sont trois fois plus importantes que celles utiles pendant le développement. C'est pourquoi un intérêt croissant apparait pour les mesures quantitatives qui permettent de donner des informations relatives aux phases de maintenance, de développement,... Les renseignements les plus importants à obtenir sont les ressources utiles et le temps nécessaire pour la réalisation de chacune des phases.

Trois utilisations importantes se distinguent.

Ces informations peuvent, tout d'abord être utilisées pour mesurer la difficulté des programmeurs à comprendre un programme et la rapidité d'implémentation de modifications. Plusieurs métriques peuvent être mises en corrélation avec la difficile expérience des programmeurs dans le travail de maintenance. Une estimation de la puissance des personnes nécessaires peut être faite ainsi qu'une évaluation du nombre de personnes nécessaires.

Les informations peuvent, dans un second temps, être utilisées comme feedback pour les utilisateurs durant le développement, indiquant les problèmes potentiels pouvant survenir.

Une troisième utilisation consisterait à fournir un guide dans les phases de tests.

Les complexités de Halstead et McCabe présentées jusqu'à présent sont très utilisées pour fournir de telles informations.

Il faut cependant distinguer la complexité calculable de la complexité psychologique. La première se référe aux aspects quantitatifs des solutions apportées aux problèmes implémentables. La deuxième se rapporte aux caractéristiques du software qui font que celui-ci est difficile à comprendre et à utiliser. La complexité calculable est la difficulté de vérifier la correction d'un algorithme. La complexité psychologique, quant à elle, se rapporte aux performances humaines.

Examinons comment les mesures de complexité peuvent nous aider dans trois domaines:
- la maintenance
- la discipline du programmeur
- la correction des erreurs.

2. Les tâches de maintenance

Dans des articles de 1979, Curtis, Sheppard, Milliman et Love [5, 6] étudient les extensions des mesures de Halstead et McCabe pour la compréhension et la modification de programmes existants. Les auteurs réalisent deux expériences.

2.1 Les expériences

La première consiste à étudier la compréhension de programmes existants et la deuxième étudie l'implémentation de modifications sur de tels programmes.

Ces expériences vont être réalisées par 36 programmeurs. Les participants au premier test ont une expérience professionnelle du Fortran depuis plus ou moins

7 ans. Quant à ceux de la seconde, ils ont une expérience du Fortran inférieure (plus ou moins 5 ans).

Les deux groupes vont analyser des programmes appartenant à trois catégories: scientifiques, statistiques et non numériques. Trois programmes de chacune des classes vont être utilisés.

Expérience 1

Dans la première expérience, les participants vont étudier pendant 25 minutes trois programmes, puis reconstruire de mémoire des formes équivalentes. Pour cette étape, ils ne disposent que de 20 minutes. La variable étudiée est le nombre d'instructions correctement reconstruites.

Expérience 2

Dans la deuxième expérience, les participants auront un temps illimité pour réaliser trois modifications sur trois programmes différents. Dans ce cas, les variables observées sont la justesse des modifications et le temps nécessaire pour les effectuer.

2.2 Les résultats

Expérience 1

Pour la première expérience, une moyenne de 51 pour cent des instructions ont été correctement reconstruites. Les performances sont très différentes pour les trois sortes de programmes. Pour chaque problème, trois flux de contrôle sont définis. De plus, trois groupes de variables sont manipulés. Un premier groupe est constitué des variables dont les noms sont trés significatifs, un second est formé des variables dont les noms les plus fréquents sont significatifs. Enfin, le dernier est constitué des variables dont les noms sont formés de un ou

deux caractères seulement. Au total, l'expérience comprend donc 108 programmes. Curtis, Sheppard, Milliman et Love séparent les données en deux catégories: les données agrégées et les données non agrégées. Les premières proviennent du premier groupe, les autres provenant des groupes restants. Pour ces deux catégories, les corrélations avec trois mesures sont présentées dans le tableau 6.1.

Tableau 6.1

Type de variables	corrélations E	v(G)	longueur
données non agrégées	-.19	-.34	-.47
données agrégées	-.13	-.35	-.53

Ces corrélations sont toutes négatives, indiquant que peu de lignes sont reconstruites correctement si le niveau de complexité (les trois mesures) augmente. On observe de légères différences entre les données des deux premiers types. La longueur et le nombre cyclomatique de McCabe sont peu en rapport avec les performances (entre 30 et 53 pour cent),tandis que la relation observée avec la complexité de Halstead est pratiquement inexistante (10 pour cent seulement).

Expérience 2

Dans cette deuxième expérience, trois sortes de commentaires sont utilisés: global, dans la ligne et sans commentaire. Les commentaires globaux sont placés aux débuts des programmes et fournissent des informations relatives sur les fonctions et les variables utilisées. Les commentaires en ligne sont entremélés dans le programme avec les instructions. Ils décrivent les fonctions spécifiques aux sections. De nouveau, trois niveaux de flux de contrôle sont envigagés, ce qui de nouveau donne 108 programmes différents à analyser. Les corrélations entre les trois mesures de complexité et les deux sortes de variables déjà rencontrées sont présentés dans le tableau 6.2.

Tableau 6.2

Critères	Corrélations E	v(G)	longueur
- Non agrégé			
. Exactitude			
prog . non modifiés	-.12	-.21	-.17
prog. modifiés	-.17	-.21	-.20
. Temps de réalisation			
prog. non modifiés	.16	.15	.13
prog. modifiés	.28	.24	.30
- agrégé			
. Exactitude			
prog. non modifiés	-.21	-.36	-.28
prog. modifiés	-.29	-.36	-.34
. Temps de réalisation			
prog. non modifiés	.25	.23	.20
prog. modifiés	.44	.38	.46

On remarquera que les coéfficients calculés avec les arguments agrégés sont plus élevés que ceux réalisés à l'aide de données non agrégées. L'interaction entre les mesures de complexité et le temps de traitement est plus forte que celle réalisée entre complexité et exactitude, spécialement avec les programmes modifiés.

2.3 Les conclusions

Les deux expériences consistaient en l'étude de la relation entre les mesures de complexité et la difficulté des programmeurs à comprendre et à modifier des softwares existants. Les performances données par Curtis, Sheppard, Milliman et Love sont moins bonnes que celles données par Halstead[18]. La raison peut être le fait que les mesures faites par ce dernier le sont sur des programmes plus importants.

Les métriques fournissent plus d'informations sur les

programmes non structurés ou non commentés. En ce qui concerne les programmes structurés et commentés, les informations disponibles altèrent la complexité psychologique à tel point que celles-ci ne sont pas reflétées sur les mesures de complexité.

Les résultats suggèrent que les mesures de complexité donnent des informations concernant les besoins humains et les coûts. Il faut aussi distinguer les interactions entre les caractéristiques des programmes et l'expérience du programmeur. Les observations montrent que les métriques sont beaucoup plus une image des performances des programmeurs moins expérimentés que des autres. Ceci amène un rapport entre bons et pauvres programmeurs de l'ordre de 28 contre 1. Ce rapport est très important, il signifie que l'entrainement, la sélection et le "placement" des programmeurs sont importants.

Il apparait finalement que les métriques fournissent des informations de base nécessaires aux tâches de maintenance, mais que de plus amples mesures ne sont possibles qu'en requérant des éléments autres que les nombres de Halstead (n_1,n_2,N_1,N_2) et les chemins élémentaires de McCabe.

Certaines différences entre programmes jouent un rôle important dans ces expériences. Les mesures de Halstead et McCabe donnent des informations sur ces différences, mais il existe d'autres facteurs non repris par ces métriques influençant la complexité psychologique.

3. Etude du style de programmation

En 1979, Gordon proposa un article [16,17] dans lequel il étudiait la clareté d'un programme estimant que cette mesure était importante pour la production de software et le développement de langage.Cette étude allait lui inspirer les mesures de complexité "pures" déjà étudiées dans un des chapitres précédents. Mais allons plus

loin, étudions l'utilité d'une telle théorie.

Une amélioration du style, du langage apporte à long terme une diminution de l'effort nécessaire dans la compréhension et la maintenance de softwares. La seule chose qui est donc à réaliser, est la minimisation de l'effort mental. Plusieurs choses influencent la difficulté de compréhension: la capacité du programmeur, la forme du programme et sa structure. Plusieurs personnes ont étudié ces problèmes. Il en ressort que la facilité d'utilisation, la connaissance approfondie d'un langage facilite la compréhension d'utilisateurs autres que le concepteur. La familiarité du programmeur avec le domaine dans lequel l'application doit être développée joue aussi un rôle très important. Mais le plus important est de réaliser une métrique qui est une simple fonction de la structure du programme et non influencée par le programmeur. Cette métrique est celle proposée par Gordon.

Cette étude permet ainsi la comparaison de plusieurs programmes de domaines différents écrits par des personnes de compétence différente. Elle fournit également un outil pour la comparaison des différences entre plusieurs applications.

Plusieurs facteurs influencent la clareté d'un programme. On peut citer parmis ceux-ci les nombres importants de techniques de construction d'algorithmes, de transformations et de guides à la programmation. Il est cependant impossible selon Gordon d'étudier tous les facteurs.

Les recherches réalisées démontrent cependant qu'il est possible d'estimer l'effort nécessaire pendant la construction d'un programme comme la mesure de la clareté de ce dernier. Quant un programmeur comprend ce que le programme réalise, il est aisé pour lui de le traduire dans un langage de programmation particulier. Dans ce cas, le temps de programmation et le temps de compréhension sont à peu près égaux. Beaucoup d'auteurs ont proposé des règles à respecter pour obtenir des programmes plus faciles à comprendre. Voici celles données par Gordon:

1) ne pas calculer la même valeur plus d'une fois
2) ne pas insérer des instructions qui ne seront jamais exécutées
3) maintenir la même signification des variables

4) éviter de constituer des IF avec une clause THEN nulle
5) éliminer les variables intermédiaires non nécessaires
6) utiliser un GOTO seulement pour sortir d'une structure itérative.

Les études apportent également un certain nombre de coéfficients de corrélation entre temps de programmation mesuré et temps calculé (tableau 6.3).

Tableau 6.3

Fortran	0.87
PL/I	0.94
APL	0.93.

Ces résultats sont obtenus par étude de 36 programmes (12 écrits dans chacun des langages).

4. Estimation du nombre d'erreurs

On peut dire aujourd'hui que le nombre d'erreurs dans un programme joue un rôle important. C'est pourquoi un certain nombre de personnes ont effectué des recherches dans ce domaine. On peut estimer que dans 60 ou 100 lignes de code, une seule erreur est présente. Mais cela n'est pas encore une assez bonne estimation. De nouveau, la quantité d'erreurs dépend d'un nombre important de facteurs comme l'expérience du programmeur, la méthode de programmation, le temps de disponibilité machine, ...

4.1 Halstead

Halstead[18] a développé en même temps que sa mesure de complexité une estimation du nombre d'erreurs dans un programme. Rappelons cette évaluation:

$$B = \frac{E}{E_o} \qquad (6.1)$$

où E_o représente le nombre moyen de discriminations entre deux erreurs. Il faut aussi noter que la complexité E est aussi une bonne estimation puisque le coéfficient de corrélation avec le nombre d'erreurs égale 0.98.

4.2 Klobert

Klobert arrive, suivant d'autres hypothéses à une formule différente. Examinons cette dernière en détail.

En utilisant la notation introduite par Halstead, le nombre de discriminations mentales E est donné en (1.10):

$$E = \frac{V}{L} \qquad (6.2)$$

ou encore

$$E = \frac{n_1 N_2 (N_1 + N_2) \log_2(n_1 + n_2)}{2 n_2} \qquad (6.3)$$

Dans le but d'une normalisation, Klobert affirme que chaque instruction du programme contient un opérateur et

un opérande. Chaque instruction est donc un appel de procédure avec un seul paramètre. Donc, pour un programme de 1000 instructions, N_1 et N_2 égalent 1000. Si on réalise la substitution dans la formule précédante, on observe que $N_1 \times N_2$ donne une puissance de 10, ce qui affecte la position de la virgule et donc peut être ignorée. On obtient:

$$E = \frac{n_1 \log_2(n_1 + n_2)}{n_2} \qquad (6.4)$$

On peut observer sur cette formule une relation entre E et le nombre d'opérateurs distincts pour un nombre donné d'opérandes distincts. Si ce dernier pourcentage est inférieur à 20, alors E est une fonction exponentielle du nombre d'opérateurs distincts. D'un autre côté, si ce nombre est plus grand que 20 pour cent, on obtient une relation linéaire.

Ce résultat fournit une bonne estimation du nombre d'erreurs. L'équation est la suivante:

$$\text{Taux d'erreurs} = E_r = \frac{n_1 \log_2(n_1 + n_2)}{n_2\ 1000^3} \qquad (6.5)$$

où n_1, n_2 sont les pourcentages décimaux d'opérandes distincts et d'opérateurs distincts multipliés par 1000.

Pour utiliser cette formule dans un environnement réel, on doit d'abord obtenir une copie représentative du code dans le système. Ensuite, les nombres U_o, U_r, V_o et V_r sont calculés comme suit:

U_o = 64 + le nombre de données transférées
V_r = (le nombre d'instructions x le nombre moyen de référence aux données) par instructiuon
$V_o = V_r$
U_r = 2 x nombre de déclarations de données.

Klobert fournit également une formulation de ces quatre nombres en fonction de l'estimation de n_1 et n_2:

$$n_1^* = (U_o / V_o)\ 10^3 \qquad (6.6)$$

$$n_2^* = (U_r / V_r)\ 10^3 \ . \qquad (6.7)$$

A partir de ces relations, on peut calculer le taux d'erreurs et pour un programme de longueur donnée, le nombre total d'erreurs devient le suivant:

E_r = longueur x le taux d'erreurs x 0.8. (6.8)

Le 0.8 provient du fait que 80 pour cent des erreurs théoriques sont anticipées.

Il est important de remarquer que E_r est le nombre d'erreurs réalisées pendant la création du module. La localisation et la correction des erreurs ne réduisent pas E_r à 0. E_r ne peut donc pas être utilisé pour indiquer la fin de la phase de test.

4.3 Ottenstein

Linda Ottenstein[27] comme nous l'avons déjà vu dans le deuxième chapître, a présenté un modèle destiné à estimer le nombre d'erreurs restant dans le système au commencement des tests et des phases de développement. Elle aboutit à la formule suivante:

$$B_v^* = \frac{V}{E_o} \ . \qquad (6.9)$$

En partant d'étude sur le nombre d'informations que la mémoire à court terme d'un individu peut retenir, on peut déduire une estimation du nombre d'opérandes en input et output. En connaissant ce nombre égale à 6, Ottenstein développe le volume V:

$$V^* = (2 + n_2^*) \log_2(2 + n_2^*) = 8 \log_2 (8) = 24 . \qquad (6.10)$$

En utilisant la valeur de lamda pour l'anglais, c'est à dire 2.16, on obtient par substitution dans la formule (1.11) le résultat suivant:

$$E_o = \frac{24^3}{2.16^2} \cong 3000 . \qquad (6.11)$$

Cela implique que après environ 3000 discriminations mentales, une décision a été prise. Le résultat de celle-ci, correct ou incorrect, est utilisé pour l'opération suivante comme entrée de l'environnement. Cela entraîne donc bien un nombre de discriminations entre deux manifestations d'une erreur égale à plus ou moin 3000. On obtient donc une estimation du nombre d'erreurs:

$$B_v^* = \frac{V}{3000} . \qquad (6.12)$$

Ottenstein développe également une estimation du temps nécessaire pour déterminer une erreur pendant la phase de validation d'un projet. Ce temps peut être obtenu à partir de la formule (1.12) par approximation:

$$T_v = \frac{K\ T}{B_v^*} . \qquad (6.13)$$

K est une estimation de la proportion du programme que le programmeur a besoin d'examiner pour découvrir l'erreur. En substituant le temps total et le nombre d'erreurs par les valeurs déjà rencontrées, on obtient

$$T_v = K \frac{E\ /\ S}{V\ /\ 3000} \qquad (6.14)$$

ou encore

$$T_v = K \frac{3000}{L\ S} \quad . \qquad (6.15)$$

Pour utiliser ce modèle, il faut disposer d'une valeur pour K. Celle-ci peut être obtenue par les caractéristiques de K. Au maximun, le temps pour découvrir toutes les erreurs et les corriger est le temps de validation. Cela représente plus ou moins 40 pour cent du temps d'implémentation, cela implique que K est inférieur à 0.40. Cette valeur donne une assez bonne estimation.

Un autre modèle calculant le nombre d'exécutions nécessaires pour valider le programme peut être construit à partir du nombre estimé d'erreurs. Pour chaque erreur détectée, une exécution en moyenne sera nécessaire, une autre étant utile pour montrer la suppression de l'erreur. Ceci implique que le nombre d'exécutions soit plus ou moins deux fois le nombre de fautes. On obtient donc la formule suivante:

$$R_v = 2\ B_v^* \quad . \qquad (6.16)$$

En estimant que la validation représente 40 pour cent du temps d'implémentation total, une estimation du nombre d'exécutions par jour peut être calculée. Ceci donne:

$$R_v\ /\ jour = \frac{2\ B_v^*}{0.4\ T} \qquad (6.17)$$

ou encore aprés substitution,

$$R_v\ /\ jour = 48\ S\ L \quad . \qquad (6.18)$$

Cette dernière formule montre que le nombre d'exécutions par jour est uniquement fonction du niveau d'implémentation du projet et de la vitesse du programmeur.

Ce nombre est très important non pour le programmeur mais bien pour le chef de projet. La somme pour tous les projets en cours du nombre d'exécutions nécessaires par jour donne une bonne mesure du taux d'occupation, de la

charge de travail.

Lorsque toutes les erreurs ont été détectées, le programme doit normalement être sûr, mais il se fait qu'il n'existe aucune méthode donnant le nombre exact de fautes. On ne connait qu'une approximation de celui-ci. Si le nombre d'erreurs trouvé est inférieur à celui calculé, il faut prendre attention avant de déclarer le programme fiable.

CHAPITRE 7:

LES CONCLUSIONS

CONCLUSION

De nombreux auteurs ont développé des théories relatives aux mesures de "qualités" des programmes. Mais, toutes ces dernières sont construites à partir des métriques trouvées par deux pionniers: Halstead et McCabe. Ses mesures sont très faciles à calculer puisque toutes celles qui ont pour origine Halstead partent de la connaissance des quatres nombres n_1,n_2,N_1 et N_2,les autres ont besoin du calcul du nombre de points décisions.

Cependant, un outil est nécessaire pour réaliser cela rapidement. L'analyseur syntaxique est dans ce but un outil très rapide, mais sa mise en oeuvre est relativement compliquée. En effet, la syntaxe du formalisme dans lequel les programmes sont écrits, doit tout d'abord être introduite et ensuite transformée. De plus, la documentation n'est pas toujours des plus facile à utiliser. Ces deux étapes sont très coûteuses en temps. Lorsqu'elles sont réalisées, le reste est très rapide.

Grâce à cet automate de mesure, il est facile de calculer les complexités de Halstead et McCabe. Mais, certaines imperfections dues à la syntaxe introduite apparaissent. On peut citer par exemple les entiers négatifs décomposés en deux parties: le signe et ensuite le nombre, les génériques dans lesquels le caractère " " n'est pas admis,... Cela pourrait être corrigé moyennant une transformation complète de la syntaxe.

Dans le cadre de ce travail, le langage pascal a été étudié, mais il est possible grâce à l'analyseur syntaxique d'utiliser tous les formalismes du basic au français, à l'anglais. Le domaine d'application de cette discipline est donc très vaste.

L'examen d'une tâche plus importante apporterait également des renseignements importants sur l'interêt de la discipline, mais cela demanderait des modifications importantes du texte original dans le but de respecter la syntaxe introduite.

Il serait également possible de développer des mesures dynamiques car toutes celles qui sont présentées dans ce mémoire sont statiques. Ces mesures seraient utilisées lors

de l'exécution, par exemple, pour compter le nombre de passages dans une partie,le nombre d'exécutions d'une instruction,... Cela serait possible par exécution simulée.

La discipline de "Software metrics" devrait avoir un bon avenir si celle-ci était plus utilisée. En effet, un grand nombre de théories ont été développées car les chercheurs se sont rendu compte de l'importance des mesures réalisées sur les programmes. Grâce à celles-ci, les utilisateurs peuvent connaitre la qualité des produits, les programmeurs peuvent avoir une estimation du nombre d'erreurs, du temps nécessaire pour les corriger, du temps nécessaire à l'implémentation, de la charge de travail utile,... Ces informations peuvent être utilisées dans des domaines différents comme la maintenance, les tests, le développement,...

Les mesures de complexité sont donc importantes. Il est à regretter qu'elles ne soient pas plus utilisées.

CHAPITRE 8:

ANNEXES

ANNEXES

1. La syntaxe Pascal (fichier S1) et README
2. Le fichier GCABST.TEXT
3. Le fichier GCLOI
4. Le fichier DEF.C
5. Le fichier de compilation MAKEFILE
6. Le texte du programme principal (FMEMTXT.C)
7. Les exemples
   - Les programmes des reines (STEST3,STEST31)
   - La transformation du "while" en "for" (TT1,TT1-1)
   - La transformation du "if+else" en "if" (TT2,TT2-1)
   - L'utilisation de procédures (TT3 à TT7,TT3-1 à TT7-1)

CHAPITRE 9:

LES REFERENCES

REFERENCES

[1] : A.L.BAKER, S.H.ZWEBEN
"The Use of Software Science in Evaluating Modularity Concepts"
IEEE Transactions on Software Engineering
vol. 5, n° 2, march 1979, pp. 110-120

[2] : A.L.BAKER, S.H.ZWEBEN
"A Comparaison of Measures of Control Flow Complexity"
IEEE Transactions on Software Engineering
vol. 6, n° 6, november 1980, pp. 506-512

[3] : M.BUYSE,...
Documentation concernant l'analyseur syntaxique
october 1985

[4] : M.L.COOK
"Software Metrics: An Introduction and Annoted Bibliography"
Software Engineering Notes
vol. 7, n° 2, april 1982, pp. 41-60

[5] : B.CURTIS, S.SHEPPARD, P.MILLIMAN, M.A.BORST, T.LOVE
"Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics"
IEEE Transactions on Software Engineering
vol., 5 n° 2, march 1979, pp. 96-104.

[6] : B.CURTIS, S.SHEPPARD, P.MILLIMAN
"Third Time Charm: Stronger Prediction pf Programmer Performance by Software Complexity Metrics"
4th International Conference on Software Engineering
september 1979, pp. 356-360.

[7] : E.W.DIJKSTRA
"Goto Statement Considered Harmful"
Communication of ACM
vol. 11, 1968, pp. 147-148

[8] : V.DONZEAU-GOUGE, G.KAHN, B.LANG, G.HUET
"Programming Environments Based on Structured Editor: the MENTOR Experience"
Rapport de Recherche INRIA
n° 26, 1980

REFERENCES

[9] : L.O.EJIOGU
"A Simple Measure of Software Complexity"
Sigplan Notices
vol. 20, nº 3, march 1985, pp. 16-31

[10] : J.L.ELSHOFF
"Measuring Commercial PL/I Programs Using Halstead's Criteria"
Sigplan Notices
vol. 11, Nº 5, may 1976, pp. 38-46

[11] : J.L.ELSHOFF
"An Analysis of Some Commercial PL/I Programs"
IEEE Transactions on Software Engineering
vol. 2, nº 2, june 1976, pp. 113-120

[12] : J.L.ELSHOFF
"An Investigation into the Effects of the Counting Method Used on Software Science Measurements"
Sigplan Notices
vol. 13, nº 2, february 1978, pp. 30-45

[13] : J.L.ELSHOFF
"A Study of the Structural Composition of PL/I Programs"
Sigplan Notices
vol. 13, nº 6, june 1978, pp 29-37

[14] : A.FITZSIMMONS, T.LOVE
"A review and Evaluation of Software Science"
Computing Surveys
vol. 10, nº 1, march 1978, pp. 3-18

[15] : T.GILB
"Software metrics"
1977

[16] : R.D.GORDON
"Measuring Improvements in Program Clarity"
IEEE Transactions on Software Engineering
vol. 5, nº 2, march 1979, pp. 79-90

REFERENCES

[17] : R.D.GORDON
"A Qualitative Justification for a Measure of Program Clarity"
IEEE Transactions on Software Engineering
vol. 5, nº 2, march 1979, pp. 121-128

[18] : M.HALSTEAD
"Elements of Software Science"
1977

[19] : W.J.HANSEN
"Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)"
Sigplan Notices
vol. 13, nº 3, march 1978, 00. 29-33

[20] : W.A.HARRISON, K.I.MAGEL
"A Topological Analysis of the Complexity of Computer Programs with less than Three Binary Branches"
Sigplan Notices
vol. 16, nº 4, 1981, pp. 51-63

[21] : W.A.HARRISON, K.I.MAGEL
"A Complexity Measure Based on Nesting Level"
Sigplan Notices
vol. 16, nº 3, 1981, pp. 63-74

[22] : T.C.JONES
"Measuring Programming Quality and Productivity"
IBM Systems Journal
vol. 17, nº 1 , 1978, pp. 39-63

[23] : K.W.KOLENCE
"An Introduction to Software Physics"
1976

[24] : T.J.McCABE
"A Complexity Measure"
IEEE Transactions on Software Engineering
vol. 2, nº 4, december 1976, pp. 308-320

[25] : S.N.MOHANTY
"Models and Measurements for Quality Assessment of Software"
Computing Surveys
vol. 11, nº 3, september 1979, pp. 251-275

REFERENCES

[26] : G.J.MYERS
"An Extension to the Cyclomatic Measure of Program Complexity"
Sigplan Notices
vol. 12, nº 10, october 1977, pp. 61-64

[27] : L.OTTENSTEIN
"Quantitative Estimates of Debugging Requirements"
IEEE Transactions on Software Engineering
vol. 5, nº 5, september 1979, pp. 504-514

[28] : A.SHROEDER
"Integrated Program Measurement and Documentation Tools"
IEEE Transactions on Software Engineering
1984

[29] : L.WEISSMAN
"Psychological Complexity of Computer Programs: An Experimental Methodology"
Sigplan Notices
vol. 9, nº 6, june 1974, pp. 25-36

en dernière minute:

L.A.ARTHUR
"Measuring Programming Productivity and Software Quality"
iley-Interscience Publication
1985

FACULTES UNIVERSITAIRES N.D DE LA PAIX
NAMUR

Institut d'informatique

Année académique 1985-1986

MESURES DE COMPLEXITE DE PROGRAMMES PASCAL

PROMOTEUR:

J.RAMAEKERS

Mémoire présenté
pour l'obtention
du grade de licencié
et maitre en
informatique par
GREGOIRE christophe

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  +                                                                            +
  +                              FICHIER S1                                    +
  +                                                                            +
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/* Ce fichier contient la syntaxe du formalisme GC ecrite en LDF.
 *
 * Pour transformer ce fichier grace a JTransfo, tout commentaire doit etre
 * elimine, ce qui implique une copie de ce fichier et une suppression
 * de tout texte parasite.
 *                                                                           */
```

```
(* ldf *)
definition de gc;


<point_d_entree> ::=    <program>;

<program> ::=   <tete_prog> &
                <bloc> "." ;

<tete_prog> ::= "program" <ident> ";" ;

SC<bloc> ::=    <int0_01> | <int0_02> | <int0_03> | <int0_04> ;
<int0_01> ::=   <bloc_1> <part_decl_prfo> <part_inst> ;
<int0_02> ::=            <part_decl_prfo> <part_inst> ;
<int0_03> ::=   <bloc_1>                  <part_inst> ;
<int0_04> ::=                             <part_inst> ;


SC<bloc_1> ::=  <int1_01> | <int1_02> | <int1_03> | <int1_04> | <int1_05> |
                <int1_06> | <int1_07> | <int1_08> | <int1_09> | <int1_10> |
                <int1_11> | <int1_12> | <int1_13> | <int1_14> | <int1_15> ;

<int1_01> ::=   <decl_etiq> <decl_const> <decl_type> <decl_var> ;
<int1_02> ::=               <decl_const> <decl_type> <decl_var> ;
<int1_03> ::=   <decl_etiq>              <decl_type> <decl_var> ;
<int1_04> ::=   <decl_etiq> <decl_const>             <decl_var> ;
<int1_05> ::=   <decl_etiq> <decl_const> <decl_type>            ;
<int1_06> ::=                            <decl_type> <decl_var> ;
<int1_07> ::=               <decl_const>             <decl_var> ;
<int1_08> ::=               <decl_const> <decl_type>            ;
<int1_09> ::=   <decl_etiq>                          <decl_var> ;
<int1_10> ::=   <decl_etiq>              <decl_type>             ;
<int1_11> ::=   <decl_etiq> <decl_const>                         ;
<int1_12> ::=   <decl_etiq>                                      ;
<int1_13> ::=               <decl_const>                         ;
<int1_14> ::=                            <decl_type>             ;
<int1_15> ::=                                        <decl_var> ;


<decl_etiq> ::=         <suit_etiq> ;
<suit_etiq> ::=         "label" <suit_etiq_1> ";" @ ;
L<suit_etiq_1> ::=      <etiq>+ "," ;

<decl_const> ::=        <suit_decl_const> ;
<suit_decl_const> ::=   "const" <suit_const> @ ;
L<suit_const> ::=       <def_const>+ # ;
<def_const> ::=         <ident> "=" <const> ";" ;

<decl_type> ::=         <suit_decl_type> ;
<suit_decl_type> ::=    "type" <suit_type> @ ;
L<suit_type> ::=        <def_type>+ # ;
<def_type> ::=          <ident> "=" <type> ";" ;
```

```
SC<type> ::=      <type_simple> |
                  <type_struct> |
                  <type_point> ;
SC<type_simple> ::=       <type_scal> |
                          <type_inte> |
                          <ident> ;
<type_scal> ::= "(" <suit_ident> ")" ;
L<suit_ident> ::=         <ident>+ "," ;
<type_inte> ::= <const> ".." <const> ;
SC<type_struct> ::=       <type_struct_np> |
                          <type_struct_p> ;
<type_struct_p> ::=       "packed" <type_struct_np> ;
SC<type_struct_np> ::=    <type_tabl> |
                          <type_arti> |
                          <type_ense> |
                          <type_fich> ;
<type_tabl> ::= "array [" <list_indice> "] of" <type> ;
L<list_indice> ::=        <type_simple>+ "," ;
<type_arti> ::=           "record" # <list_cham> # "end" ;
SC<list_cham> ::=         <part_fixe> |
                          <part_mixe> |
                          <part_variante> ;
L<part_fixe> ::=          <defi_champs>+ ";" # ;
SC<defi_champs> ::=       <defi_champs_1> | <vide> ;
<defi_champs_1> ::=       <suit_ident> ":" <type> ;
<part_mixe> ::=           <part_fixe> ";" # <part_variante> ;
SC<part_variante> ::=     <int2_01> | <int2_02> ;
<int2_01> ::=             "case" <sele_champ> <ident> "of" # <variante_list> ;
<int2_02> ::=             "case"                <ident> "of" #  <variante_list> ;
L<variante_list> ::=      <variante>+ ";" # ;
SC<variante> ::=                <variante_1> | <vide> ;
SC<variante_1> ::=        <int4_01> | <int4_02> | <int4_03> ;
<int4_01> ::=             <list_etiq_arti> ": (" <part_fixe> ")" ;
<int4_02> ::=             <list_etiq_arti> ": (" <part_mixe> ")" ;
<int4_03> ::=             <list_etiq_arti> ": (" <part_variante> ")" ;
L<list_etiq_arti> ::=     <const>+ "," ;
<sele_champ> ::=          <type_iden> ;
<type_iden> ::=           <ident> ":" ;
<type_ense> ::=           "set of" <type_simple> ;
<type_fich> ::=           "file of" <type> ;
<type_point> ::=          "^" <ident> ;

<decl_var> ::=            <part_decl_var> ;
<part_decl_var> ::=       "var" <decl_var_1> @ ;
L<decl_var_1> ::=         <decl_var_2>+ # ;
<decl_var_2> ::=          <suit_ident> ":" <type> ";" ;



<part_decl_prfo> ::= <fonct_proc> ";" @ ;

L<fonct_proc> ::= <decl_prfo>+ ";" @ ;

SC<decl_prfo> ::= <decl_fonct> |
                  <decl_proc> ;
```

```
SC<decl_proc> ::= <decl_proc_1> ;
<decl_proc_1> ::= <en_tete_proc> 2 <bloc> ;
SC<en_tete_proc> ::= <en_tete_proc_1> | <en_tete_proc_2> ;
<en_tete_proc_1> ::= "procedure" <ident> ";" ;
<en_tete_proc_2> ::= "procedure" <ident> "(" <list_list> ") :" ;
SC<list_list> ::= <group_para> | <var_group> | <funct_group> | <proc_group> ;
<group_para> ::= <suit_ident> ":" <ident> ;
<var_group> ::= "var" <group_para> ;
<funct_group> ::= "function" <group_para> ;
<proc_group> ::= "procedure" <suit_ident> ;


SC<decl_fonct> ::= <decl_fonct_1> ;
<decl_fonct_1> ::= <en_tete_fonct> 2 <bloc> ;
SC<en_tete_fonct> ::= <en_tete_fonct_1> | <en_tete_fonct_2> ;
<en_tete_fonct_1> ::= "function" <ident> ":" <ident> ";" ;
<en_tete_fonct_2> ::= "function" <ident> "(" <list_list> ") :" <ident> ";" ;


<part_inst> ::= "begin" + <list_inst> 3 "end" ;
L<list_inst> ::= <inst>+ ";" 3 ;

SC<inst> ::=    <inst_etiq> |
                <inst_non_etiq> ;
SC<inst_etiq> ::= <inst5_01> | <inst5_02> ;
<inst5_01> ::=   <etiq> ":" <inst_non_etiq> ;
<inst5_02> ::=   <etiq> ":" ;
SC<inst_non_etiq> ::= <inst_simple> |
                <inst_stru> ;

SC<inst_simple> ::= <inst_affec> |
                <inst_appe> |
                <inst_bran> ;

<inst_affec> ::= <vari> ":=" <expr> ;
ALT<inst_appe> ::= <ident> | <rec_appel> ;
<inst_bran> ::= "goto" <etiq> ;

SC<inst_stru> ::= <part_inst> |
                  <inst_sele> |
                  <inst_iter> |
                  <inst_avec> ;

SC<inst_sele> ::= <inst_si> | <inst_cas> ;
SC<inst_si> ::=  <if_then> | <if_then_else> ;
<if_then> ::= "if" <expr> + "then" <inst> ;
<if_then_else> ::= "if" <expr> + "then" <inst> + "else" <inst> ;
<inst_cas> ::= "case" <expr> "of" # <test_elem_list> $ "end" ;
L<test_elem_list> ::= <elem_list_cas>+ ";" $;
SC<elem_list_cas> ::= <list_etiq_cas_1> ;
<list_etiq_cas_1> ::= <list_etiq_cas> ":" <inst>;
L<list_etiq_cas> ::= <const>+ "," ;
```

```
SC<inst_iter> ::= <inst_tant_que> | <inst_repe> | <inst_pour> ;
<inst_tant_que> ::= "while" <expr> "do" # <inst> ;
<inst_repe> ::= "repeat" # <list_inst> # "until" <expr> ;
<inst_pour> ::= "for" <inst_affec> <list_pour> "do" # <inst> ;
SC<list_pour> ::= <list_pour_1> | <list_pour_2> ;
<list_pour_1> ::= "to" <expr> ;
<list_pour_2> ::= "downto" <expr> ;
<inst_avec> ::= "with" <list_vari_arti> # "do" # <inst> ;
L<list_vari_arti> ::= <vari>+ "," ;

SC<vari> ::= <ident> |
             <vari_sans_ident> ;
SC<vari_sans_ident> ::= <vari_indi> |
                        <desi_cham> |
                        <vari_pointee_ou_tamp_fich> ;
<vari_indi> ::= <vari> "[" <list_expr> "]" ;
L<list_expr> ::= <expr>+ "," ;
<desi_cham> ::= <vari> ^"."^ <ident> ;
<vari_pointee_ou_tamp_fich> ::= <vari> ^"^" ;

SC<fact> ::=    <ident> |
                <vari_sans_ident> |
                <cons_non_sign> |
                <rec_appel> |
                <ense> | <fact_non> | <texp> ;
SC<cons_non_sign> ::= <reel> | <etiq> | <chaine> ;
<fact_non> ::= "not" <fact> ;
<texp> ::= "(" <expr> ")" ;
<rec_appel> ::= <ident> "(" <list_para_effe> ")" ;
L<list_para_effe> ::= <para_effe>+ "," ;
SC<para_effe> ::= <expr> ;
SC<ense> ::= <ints_01> | <ints_02> ;
<ints_01> ::= "[" <list_elem> "]" ;
<ints_02> ::= "[]" ;
SC<list_elem> ::= <list_elem_1> ;
L<list_elem_1> ::= <elem>+ "," ;
SC<elem> ::= <vari_indi> | <desi_cham> | <vari_pointee_ou_tamp_fich> ;

SC<expr> ::=    <expr_simple> |
                <test_expr> ;
<test_expr> ::= <expr_simple> <oper_1> <expr_simple> ;
SC<expr_simple> ::=     <expr_oper> |
                        <term>      |
                        <term_sign> ;
<term_sign> ::=         <oper_3> <term> ;
<expr_oper> ::=         <expr_simple> <oper_2> <expr_simple> ;
SC<oper_1> ::= <oper_11> | <oper_12> | <oper_13> | <oper_14> | <oper_15> |
               <oper_16> | <oper_17> ;
<oper_11> ::= "<>" ;
<oper_12> ::= ">"  ;
<oper_13> ::= "<"  ;
<oper_14> ::= ">=" ;
<oper_15> ::= "<=" ;
<oper_16> ::= "in" ;
<oper_17> ::= "=" ;
SC<oper_2> ::= <oper_21> | <oper_22> | <oper_23> ;
```

```
<oper_21> ::= "+"  ;
<oper_22> ::= "-"  ;
<oper_23> ::= "or" ;
SC<oper_3> ::= <oper_21> | <oper_22> ;
SC<term> ::=    <fact> |
                <test_term> ;
<test_term> ::= <term> <oper_4> <term> ;
SC<oper_4> ::= <oper_41> | <oper_42> | <oper_43> | <oper_44> | <oper_45> ;
<oper_41> ::= "*"   ;
<oper_42> ::= "/"   ;
<oper_43> ::= "div" ;
<oper_44> ::= "mod" ;
<oper_45> ::= "and" ;

SC<const> ::=   <etiq_sign> |
                <reel_sign> |
                <etiq>      |
                <reel>      |
                <chaine> ;
<chaine> ::=    "'"^ <cara> ^"'" ;
SC<cara> ::=    <etiq> | <ident> ;
SC<reel_sign> ::=       <reel_1> | <reel_2> | <reel_3> ;
SC<reel> ::=            <reel_4> | <reel_5> | <reel_6> ;
<reel_1> ::= <etiq_sign> ^"."^ <etiq> ;
<reel_4> ::= <etiq>      ^"."^ <etiq> ;
<reel_2> ::= <etiq_sign> ^"."^ <etiq> ^"e"^ <fact_echel> ;
<reel_5> ::= <etiq>      ^"."^ <etiq> ^"e"^ <fact_echel> ;
<reel_3> ::= <etiq_sign> ^"e"^ <fact_echel> ;
<reel_6> ::= <etiq>      ^"e"^ <fact_echel> ;
SC<fact_echel> ::= <etiq> | <etiq_sign> ;
SC<etiq_sign> ::= <sign_etiq_1> | <sign_etiq_2> ;
<sign_etiq_1> ::= "+"^ <etiq> ;
<sign_etiq_2> ::= "-"^ <etiq> ;



<vide> ::= ;


GEN <etiq> ::= "[0-9]*";
GEN <ident> ::= "[a-zA-Z][a-zA-Z0-9_]*" .
```

Directory Transfo :

- Directory bin : contient les executables necessaires a Jtransfo
- Directory h : contient les includes
- Directory lib : contient la librairie squelette pour Jtransfo
- Directory demo : contient un programme de test
  pico.ldf
- Directory data : prevue pour l'execution de Jtransfo.

Pour la premiere fois :

```
cd Transfo ;
Jtransfo -i lib/Acom.lib data ;
```

Pour chaque fois :

```
Jtransfo nom_fichier_ldf pathname_data_directory ;
```

exemple : cd demo ;
Jtransfo pico.ldf ../data ;

pour tester : make ;
Ftest pico.txt ;

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  +                                                                            +
  +                          FICHIER GCABST.TEXT                               +
  +                                                                            +
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/*
 * Ce fichier contient les informations relatives aux noeuds donnees par
 * le transformateur.
 *                                                                          */
```

```
! ! LISTE ! ! suit_etiq_1 (1 ) :
  *   *   *   *   *   *   *   etiq   (181 )

! ! LISTE ! ! suit_const (2 ) :
  *   *   *   *   *   *   *   def_const   (43 )

! ! LISTE ! ! suit_type (3 ) :
  *   *   *   *   *   *   *   def_type   (44 )

! ! LISTE ! ! suit_ident (4 ) :
  *   *   *   *   *   *   *   ident   (185 )

! ! LISTE ! ! list_indice (5 ) :
  *   *   *   *   *   *   *   type_simple   (91 )

! ! LISTE ! ! part_fixe (6 ) :
  *   *   *   *   *   *   *   defi_champs   (98 )

! ! LISTE ! ! variante_list (7 ) :
  *   *   *   *   *   *   *   variante   (100 )

! ! LISTE ! ! list_etiq_arti (8 ) :
  *   *   *   *   *   *   *   const   (155 )

! ! LISTE ! ! decl_var_1 (9 ) :
  *   *   *   *   *   *   *   decl_var_2   (53 )

! ! LISTE ! ! fonct_proc (10 ) :
  *   *   *   *   *   *   *   decl_prfo   (110 )

! ! LISTE ! ! list_list_2 (11 ) :
  *   *   *   *   *   *   *   list_list   (114 )

! ! LISTE ! ! list_inst (12 ) :
  *   *   *   *   *   *   *   inst   (121 )

! ! LISTE ! ! test_elem_list (13 ) :
  *   *   *   *   *   *   *   elem_list_cas   (131 )

! ! LISTE ! ! list_etiq_cas (14 ) :
  *   *   *   *   *   *   *   const   (155 )
```

```
! ! LISTE ! ! list_vari_arti (15 )       :
  *   *   *   *   *   *   *   vari    (136 )

! ! LISTE ! ! list_expr (16 )    :
  *   *   *   *   *   *   *   expr    (148 )

! ! LISTE ! ! list_para_effe (17 )       :
  *   *   *   *   *   *   *   para_effe    (143 )

! ! LISTE ! ! list_elem_1 (18 ) :
  *   *   *   *   *   *   *   elem    (147 )

! ! QUATERNAIRE ! ! int1_01 (19 )        :
  *   *   *   *   *   *   *   decl_etiq    (84 )
  *   *   *   *   *   *   *   decl_const    (86 )
  *   *   *   *   *   *   *   decl_type    (88 )
  *   *   *   *   *   *   *   decl_var    (107 )

! ! TERNAIRE ! ! int0_01 (20 )   :
  *   *   *   *   *   *   *   bloc_1    (79 )
  *   *   *   *   *   *   *   part_decl_prfo    (109 )
  *   *   *   *   *   *   *   part_inst    (120 )

! ! TERNAIRE ! ! int1_02 (21 )   :
  *   *   *   *   *   *   *   decl_const    (86 )
  *   *   *   *   *   *   *   decl_type    (88 )
  *   *   *   *   *   *   *   decl_var    (107 )

! ! TERNAIRE ! ! int1_03 (22 )   :
  *   *   *   *   *   *   *   decl_etiq    (84 )
  *   *   *   *   *   *   *   decl_type    (88 )
  *   *   *   *   *   *   *   decl_var    (107 )

! ! TERNAIRE ! ! int1_04 (23 )   :
  *   *   *   *   *   *   *   decl_etiq    (84 )
  *   *   *   *   *   *   *   decl_const    (86 )
  *   *   *   *   *   *   *   decl_var    (107 )

! ! TERNAIRE ! ! int1_05 (24 )   :
  *   *   *   *   *   *   *   decl_etiq    (84 )
  *   *   *   *   *   *   *   decl_const    (86 )
  *   *   *   *   *   *   *   decl_type    (88 )

! ! TERNAIRE ! ! int2_01 (25 )   :
```

```
   *   *   *   *   *   *   *   sele_champ   (102 )
   *   *   *   *   *   *   *   ident   (135 )
   *   *   *   *   *   *   *   variante_list   (7 )

! ! TERNAIRE ! ! en_tete_fonct_2 (26 )   :
   *   *   *   *   *   *   *   ident   (135 )
   *   *   *   *   *   *   *   list_list_2   (11 )
   *   *   *   *   *   *   *   ident   (135 )

! ! TERNAIRE ! ! if_then_else (27 )      :
   *   *   *   *   *   *   *   expr   (148 )
   *   *   *   *   *   *   *   inst   (121 )
   *   *   *   *   *   *   *   inst   (121 )

! ! TERNAIRE ! ! inst_pour (28 )         :
   *   *   *   *   *   *   *   inst_affec   (60 )
   *   *   *   *   *   *   *   list_pour   (133 )
   *   *   *   *   *   *   *   inst   (121 )

! ! TERNAIRE ! ! test_expr (29 )         :
   *   *   *   *   *   *   *   expr_simple   (149 )
   *   *   *   *   *   *   *   oper_1   (150 )
   *   *   *   *   *   *   *   expr_simple   (149 )

! ! TERNAIRE ! ! expr_oper (30 )         :
   *   *   *   *   *   *   *   expr_simple   (149 )
   *   *   *   *   *   *   *   oper_2   (151 )
   *   *   *   *   *   *   *   expr_simple   (149 )

! ! TERNAIRE ! ! test_term (31 )         :
   *   *   *   *   *   *   *   term   (153 )
   *   *   *   *   *   *   *   oper_4   (154 )
   *   *   *   *   *   *   *   term   (153 )

! ! TERNAIRE ! ! reel_2 (32 )    :
   *   *   *   *   *   *   *   etiq_sign   (181 )
   *   *   *   *   *   *   *   etiq   (181 )
   *   *   *   *   *   *   *   fact_echel   (160 )

! ! TERNAIRE ! ! reel_5 (33 )    :
   *   *   *   *   *   *   *   etiq   (181 )
   *   *   *   *   *   *   *   etiq   (181 )
   *   *   *   *   *   *   *   fact_echel   (160 )

! ! BINAIRE ! ! program (34 )    :
   *   *   *   *   *   *   *   tete_prog   (76 )
   *   *   *   *   *   *   *   bloc   (77 )
```

```
! ! BINAIRE ! ! int0_02 (35 )   :
  *   *   *   *   *   *   *    part_decl_prfo   (109 )
  *   *   *   *   *   *   *    part_inst   (120 )

! ! BINAIRE ! ! int0_03 (36 )   :
  *   *   *   *   *   *   *    bloc_1   (79 )
  *   *   *   *   *   *   *    part_inst   (120 )

! ! BINAIRE ! ! int1_06 (37 )   :
  *   *   *   *   *   *   *    decl_type   (88 )
  *   *   *   *   *   *   *    decl_var   (107 )

! ! BINAIRE ! ! int1_07 (38 )   :
  *   *   *   *   *   *   *    decl_const   (86 )
  *   *   *   *   *   *   *    decl_var   (107 )

! ! BINAIRE ! ! int1_08 (39 )   :
  *   *   *   *   *   *   *    decl_const   (86 )
  *   *   *   *   *   *   *    decl_type   (88 )

! ! BINAIRE ! ! int1_09 (40 )   :
  *   *   *   *   *   *   *    decl_etiq   (84 )
  *   *   *   *   *   *   *    decl_var   (107 )

! ! BINAIRE ! ! int1_10 (41 )   :
  *   *   *   *   *   *   *    decl_etiq   (84 )
  *   *   *   *   *   *   *    decl_type   (88 )

! ! BINAIRE ! ! int1_11 (42 )   :
  *   *   *   *   *   *   *    decl_etiq   (84 )
  *   *   *   *   *   *   *    decl_const   (86 )

! ! BINAIRE ! ! def_const (43 ) :
  *   *   *   *   *   *   *    ident   (135 )
  *   *   *   *   *   *   *    const   (155 )

! ! BINAIRE ! ! def_type (44 )  :
  *   *   *   *   *   *   *    ident   (135 )
  *   *   *   *   *   *   *    type   (90 )

! ! BINAIRE ! ! type_inte (45 ) :
  *   *   *   *   *   *   *    const   (155 )
  *   *   *   *   *   *   *    const   (155 )
```

```
! ! BINAIRE ! ! type_tabl (46 ) :
  *   *   *   *   *   *   *   list_indice   (5 )
  *   *   *   *   *   *   *   type   (90 )

! ! BINAIRE ! ! defi_champs_1 (47 )     :
  *   *   *   *   *   *   *   suit_ident   (4 )
  *   *   *   *   *   *   *   type   (90 )

! ! BINAIRE ! ! part_mixe (48 ) :
  *   *   *   *   *   *   *   part_fixe   (6 )
  *   *   *   *   *   *   *   part_variante   (99 )

! ! BINAIRE ! ! int2_02 (49 )   :
  *   *   *   *   *   *   *   ident   (185 )
  *   *   *   *   *   *   *   variante_list   (7 )

! ! BINAIRE ! ! int4_01 (50 )   :
  *   *   *   *   *   *   *   list_etiq_arti   (8 )
  *   *   *   *   *   *   *   part_fixe   (6 )

! ! BINAIRE ! ! int4_02 (51 )   :
  *   *   *   *   *   *   *   list_etiq_arti   (8 )
  *   *   *   *   *   *   *   part_mixe   (48 )

! ! BINAIRE ! ! int4_03 (52 )   :
  *   *   *   *   *   *   *   list_etiq_arti   (8 )
  *   *   *   *   *   *   *   part_variante   (99 )

! ! BINAIRE ! ! decl_var_2 (53 )       :
  *   *   *   *   *   *   *   suit_ident   (4 )
  *   *   *   *   *   *   *   type   (90 )

! ! BINAIRE ! ! decl_proc_1 (54 )       :
  *   *   *   *   *   *   *   en_tete_proc   (112 )
  *   *   *   *   *   *   *   bloc   (77 )

! ! BINAIRE ! ! en_tete_proc_2 (55 )    :
  *   *   *   *   *   *   *   ident   (185 )
  *   *   *   *   *   *   *   list_list_2   (11 )

! ! BINAIRE ! ! group_para (56 )       :
  *   *   *   *   *   *   *   suit_ident   (4 )
  *   *   *   *   *   *   *   ident   (185 )
```

```
! ! BINAIRE ! ! decl_fonct_1 (57 )      :
  *   *   *   *   *   *   *   en_tete_fonct   (119 )
  *   *   *   *   *   *   *   bloc   (77 )

! ! BINAIRE ! ! en_tete_fonct_1 (58 )   :
  *   *   *   *   *   *   *   ident   (185 )
  *   *   *   *   *   *   *   ident   (185 )

! ! BINAIRE ! ! inst_01 (59 )   :
  *   *   *   *   *   *   *   etiq   (181 )
  *   *   *   *   *   *   *   inst_non_etiq   (124 )

! ! BINAIRE ! ! inst_affec (60 )      :
  *   *   *   *   *   *   *   vari   (136 )
  *   *   *   *   *   *   *   expr   (148 )

! ! BINAIRE ! ! if_then (61 )   :
  *   *   *   *   *   *   *   expr   (148 )
  *   *   *   *   *   *   *   inst   (121 )

! ! BINAIRE ! ! inst_cas (62 )   :
  *   *   *   *   *   *   *   expr   (148 )
  *   *   *   *   *   *   *   test_elem_list   (13 )

! ! BINAIRE ! ! list_etiq_cas_1 (63 )   :
  *   *   *   *   *   *   *   list_etiq_cas   (14 )
  *   *   *   *   *   *   *   inst   (121 )

! ! BINAIRE ! ! inst_tant_que (64 )      :
  *   *   *   *   *   *   *   expr   (148 )
  *   *   *   *   *   *   *   inst   (121 )

! ! BINAIRE ! ! inst_repe (65 ) :
  *   *   *   *   *   *   *   list_inst   (12 )
  *   *   *   *   *   *   *   expr   (148 )

! ! BINAIRE ! ! inst_avec (66 ) :
  *   *   *   *   *   *   *   list_vari_arti   (15 )
  *   *   *   *   *   *   *   inst   (121 )

! ! BINAIRE ! ! vari_indi (67 ) :
  *   *   *   *   *   *   *   vari   (136 )
  *   *   *   *   *   *   *   list_expr   (16 )

! ! BINAIRE ! ! desi_cham (68 ) :
```

```
   *   *   *   *   *   *   *   vari   (136 )
   *   *   *   *   *   *   *   ident   (135 )

! ! BINAIRE ! ! rec_appel (69 ) :
   *   *   *   *   *   *   *   ident   (135 )
   *   *   *   *   *   *   *   list_para_effe   (17 )

! ! BINAIRE ! ! term_sign (70 ) :
   *   *   *   *   *   *   *   oper_3   (152 )
   *   *   *   *   *   *   *   term   (153 )

! ! BINAIRE ! ! real_1 (71 )   :
   *   *   *   *   *   *   *   etiq_sign   (161 )
   *   *   *   *   *   *   *   etiq   (181 )

! ! BINAIRE ! ! real_4 (72 )   :
   *   *   *   *   *   *   *   etiq   (181 )
   *   *   *   *   *   *   *   etiq   (181 )

! ! BINAIRE ! ! real_3 (73 )   :
   *   *   *   *   *   *   *   etiq_sign   (161 )
   *   *   *   *   *   *   *   fact_echel   (160 )

! ! BINAIRE ! ! real_6 (74 )   :
   *   *   *   *   *   *   *   etiq   (181 )
   *   *   *   *   *   *   *   fact_echel   (160 )

! ! UNAIRE ! ! point_d_entree (75 )   :
   *   *   *   *   *   *   *   program   (34 )

! ! UNAIRE ! ! tete_prog (76 )  :
   *   *   *   *   *   *   *   ident   (135 )

! ! ! SC ! ! ! bloc (77 )   :
   *   *   *   *   *   *   *   int0_01   (20 )
   *   *   *   *   *   *   *   int0_02   (35 )
   *   *   *   *   *   *   *   int0_03   (36 )
   *   *   *   *   *   *   *   int0_04   (78 )

! ! UNAIRE ! ! int0_04 (78 )   :
   *   *   *   *   *   *   *   part_inst   (120 )

! ! ! SC ! ! ! bloc_1 (79 )   :
   *   *   *   *   *   *   *   int1_01   (19 )
   *   *   *   *   *   *   *   int1_02   (21 )
```

```
   *   *   *   *   *   *   *   intl_03   (22 )
   *   *   *   *   *   *   *   intl_04   (23 )
   *   *   *   *   *   *   *   intl_05   (24 )
   *   *   *   *   *   *   *   intl_06   (37 )
   *   *   *   *   *   *   *   intl_07   (38 )
   *   *   *   *   *   *   *   intl_08   (39 )
   *   *   *   *   *   *   *   intl_09   (40 )
   *   *   *   *   *   *   *   intl_10   (41 )
   *   *   *   *   *   *   *   intl_11   (42 )
   *   *   *   *   *   *   *   intl_12   (80 )
   *   *   *   *   *   *   *   intl_13   (81 )
   *   *   *   *   *   *   *   intl_14   (82 )
   *   *   *   *   *   *   *   intl_15   (83 )

! ! UNAIRE ! ! intl_12 (80 )    :
   *   *   *   *   *   *   *   decl_etiq   (84 )

! ! UNAIRE ! ! intl_13 (81 )    :
   *   *   *   *   *   *   *   decl_const   (86 )

! ! UNAIRE ! ! intl_14 (82 )    :
   *   *   *   *   *   *   *   decl_type   (88 )

! ! UNAIRE ! ! intl_15 (83 )    :
   *   *   *   *   *   *   *   decl_var   (107 )

! ! UNAIRE ! ! decl_etiq (84 )  :
   *   *   *   *   *   *   *   suit_etiq   (85 )

! ! UNAIRE ! ! suit_etiq (85 )  :
   *   *   *   *   *   *   *   suit_etiq_1   (1 )

! ! UNAIRE ! ! decl_const (86 ) :
   *   *   *   *   *   *   *   suit_decl_const   (87 )

! ! UNAIRE ! ! suit_decl_const (87 )    :
   *   *   *   *   *   *   *   suit_const   (2 )

! ! UNAIRE ! ! decl_type (88 )  :
   *   *   *   *   *   *   *   suit_decl_type   (89 )

! ! UNAIRE ! ! suit_decl_type (89 )     :
   *   *   *   *   *   *   *   suit_type   (3 )

! ! ! SC ! ! ! type (90 )       :
```

```
   *   *   *   *   *   *   *   type_simple   (91 )
   *   *   *   *   *   *   *   type_struct   (93 )
   *   *   *   *   *   *   *   type_point   (106 )


!  !  !  SC  !  !  !  type_simple (91 )        :
   *   *   *   *   *   *   *   type_scal   (92 )
   *   *   *   *   *   *   *   type_inte   (45 )
   *   *   *   *   *   *   *   ident   (185 )


!  !  UNAIRE  !  !  type_scal (92 )  :
   *   *   *   *   *   *   *   suit_ident   (4 )


!  !  !  SC  !  !  !  type_struct (93 )        :
   *   *   *   *   *   *   *   type_struct_np   (95 )
   *   *   *   *   *   *   *   type_struct_p   (94 )


!  !  UNAIRE  !  !  type_struct_p (94 )      :
   *   *   *   *   *   *   *   type_struct_np   (95 )


!  !  !  SC  !  !  !  type_struct_np (95 )     :
   *   *   *   *   *   *   *   type_tabl   (46 )
   *   *   *   *   *   *   *   type_arti   (96 )
   *   *   *   *   *   *   *   type_ense   (104 )
   *   *   *   *   *   *   *   type_fich   (105 )


!  !  UNAIRE  !  !  type_arti (96 )  :
   *   *   *   *   *   *   *   list_cham   (97 )


!  !  !  SC  !  !  !  list_cham (97 )  :
   *   *   *   *   *   *   *   part_fixe   (6 )
   *   *   *   *   *   *   *   part_mixe   (48 )
   *   *   *   *   *   *   *   part_variante   (99 )


!  !  !  SC  !  !  !  defi_champs (98 )        :
   *   *   *   *   *   *   *   defi_champs_1   (47 )
   *   *   *   *   *   *   *   vide   (180 )


!  !  !  SC  !  !  !  part_variante (99 )      :
   *   *   *   *   *   *   *   int2_01   (25 )
   *   *   *   *   *   *   *   int2_02   (49 )


!  !  !  SC  !  !  !  variante (100 )  :
   *   *   *   *   *   *   *   variante_1   (101 )
   *   *   *   *   *   *   *   vide   (180 )
```

```
! ! ! SC ! ! ! variante_1 (101 )          :
  *   *   *   *   *   *   *   int4_01    (50 )
  *   *   *   *   *   *   *   int4_02    (51 )
  *   *   *   *   *   *   *   int4_03    (52 )

! ! UNAIRE ! ! sele_champ (102 )          :
  *   *   *   *   *   *   *   type_iden    (103 )

! ! UNAIRE ! ! type_iden (103 ) :
  *   *   *   *   *   *   *   ident    (185 )

! ! UNAIRE ! ! type_ense (104 ) :
  *   *   *   *   *   *   *   type_simple    (91 )

! ! UNAIRE ! ! type_fich (105 ) :
  *   *   *   *   *   *   *   type    (90 )

! ! UNAIRE ! ! type_point (106 )          :
  *   *   *   *   *   *   *   ident    (185 )

! ! UNAIRE ! ! decl_var (107 )  :
  *   *   *   *   *   *   *   part_decl_var    (108 )

! ! UNAIRE ! ! part_decl_var (108 )      :
  *   *   *   *   *   *   *   decl_var_1    (9 )

! ! UNAIRE ! ! part_decl_prfo (109 )     :
  *   *   *   *   *   *   *   fonct_proc    (10 )

! ! ! SC ! ! ! decl_prfo (110 ) :
  *   *   *   *   *   *   *   decl_fonct    (118 )
  *   *   *   *   *   *   *   decl_proc    (111 )

! ! ! SC ! ! ! decl_proc (111 ) :
  *   *   *   *   *   *   *   decl_proc_1    (54 )

! ! ! SC ! ! ! en_tete_proc (112 )       :
  *   *   *   *   *   *   *   en_tete_proc_1    (113 )
  *   *   *   *   *   *   *   en_tete_proc_2    (55 )

! ! UNAIRE ! ! en_tete_proc_1 (113 )     :
  *   *   *   *   *   *   *   ident    (185 )
```

```
! ! ! SC ! ! ! list_list (114 ) :
  *   *   *   *   *   *   *   group_para   (56 )
  *   *   *   *   *   *   *   var_group    (115 )
  *   *   *   *   *   *   *   funct_group   (116 )
  *   *   *   *   *   *   *   proc_group    (117 )

! ! UNAIRE ! ! var_group (115 ) :
  *   *   *   *   *   *   *   group_para   (56 )

! ! UNAIRE ! ! funct_group (116 )       :
  *   *   *   *   *   *   *   group_para   (56 )

! ! UNAIRE ! ! proc_group (117 )        :
  *   *   *   *   *   *   *   suit_ident   (4 )

! ! ! SC ! ! ! decl_fonct (118 )        :
  *   *   *   *   *   *   *   decl_fonct_1   (57 )

! ! ! SC ! ! ! en_tete_fonct (119 )     :
  *   *   *   *   *   *   *   en_tete_fonct_1   (58 )
  *   *   *   *   *   *   *   en_tete_fonct_2   (25 )

! ! UNAIRE ! ! part_inst (120 ) :
  *   *   *   *   *   *   *   list_inst   (12 )

! ! ! SC ! ! ! inst (121 )      :
  *   *   *   *   *   *   *   inst_etiq   (122 )
  *   *   *   *   *   *   *   inst_non_etiq   (124 )

! ! ! SC ! ! ! inst_etiq (122 ) :
  *   *   *   *   *   *   *   int5_01   (59 )
  *   *   *   *   *   *   *   int5_02   (123 )

! ! UNAIRE ! ! int5_02 (123 )   :
  *   *   *   *   *   *   *   etiq   (131 )

! ! ! SC ! ! ! inst_non_etiq (124 )     :
  *   *   *   *   *   *   *   inst_simple   (125 )
  *   *   *   *   *   *   *   inst_stru   (128 )

! ! ! SC ! ! ! inst_simple (125 )       :
  *   *   *   *   *   *   *   inst_affec   (60 )
  *   *   *   *   *   *   *   inst_appe   (126 )
  *   *   *   *   *   *   *   inst_bran   (127 )
```

```
! ! ALTERNATIF ! ! inst_appe (126 )      :
  *   *   *   *   *   *   *   ident   (135 )
  *   *   *   *   *   *   *   rec_appel   (69 )

! ! UNAIRE ! ! inst_bran (127 ) :
  *   *   *   *   *   *   *   etiq   (131 )

! ! ! SC ! ! ! inst_stru (128 ) :
  *   *   *   *   *   *   *   part_inst   (120 )
  *   *   *   *   *   *   *   inst_sele   (129 )
  *   *   *   *   *   *   *   inst_iter   (132 )
  *   *   *   *   *   *   *   inst_avec   (66 )

! ! ! SC ! ! ! inst_sele (129 ) :
  *   *   *   *   *   *   *   inst_si   (130 )
  *   *   *   *   *   *   *   inst_cas   (62 )

! ! ! SC ! ! ! inst_si (130 )    :
  *   *   *   *   *   *   *   if_then   (61 )
  *   *   *   *   *   *   *   if_then_else   (27 )

! ! ! SC ! ! ! elem_list_cas (131 )      :
  *   *   *   *   *   *   *   list_etiq_cas_1   (63 )

! ! ! SC ! ! ! inst_iter (132 ) :
  *   *   *   *   *   *   *   inst_tant_que   (64 )
  *   *   *   *   *   *   *   inst_repe   (65 )
  *   *   *   *   *   *   *   inst_pour   (23 )

! ! ! SC ! ! ! list_pour (133 ) :
  *   *   *   *   *   *   *   list_pour_1   (134 )
  *   *   *   *   *   *   *   list_pour_2   (135 )

! ! UNAIRE ! ! list_pour_1 (134 )       :
  *   *   *   *   *   *   *   expr   (143 )

! ! UNAIRE ! ! list_pour_2 (135 )       :
  *   *   *   *   *   *   *   expr   (143 )

! ! ! SC ! ! ! vari (136 )      :
  *   *   *   *   *   *   *   ident   (135 )
  *   *   *   *   *   *   *   vari_sans_ident   (137 )

! ! ! SC ! ! ! vari_sans_ident (137 )   :
```

```
   *   *   *   *   *   *   *   vari_indi   (67 )
   *   *   *   *   *   *   *   desi_cham   (68 )
   *   *   *   *   *   *   *   vari_pointee_ou_tamp_fich   (138 )

! ! UNAIRE ! ! vari_pointee_ou_tamp_fich (138 ) :
   *   *   *   *   *   *   *   vari   (136 )

! ! ! SC ! ! ! fact (139 )      :
   *   *   *   *   *   *   *   ident   (185 )
   *   *   *   *   *   *   *   vari_sans_ident   (137 )
   *   *   *   *   *   *   *   cons_non_sign   (140 )
   *   *   *   *   *   *   *   rec_appel   (69 )
   *   *   *   *   *   *   *   ense   (144 )
   *   *   *   *   *   *   *   fact_non   (141 )
   *   *   *   *   *   *   *   texp   (142 )

! ! ! SC ! ! ! cons_non_sign (140 )     :
   *   *   *   *   *   *   *   reel   (159 )
   *   *   *   *   *   *   *   etiq   (181 )
   *   *   *   *   *   *   *   chaine   (156 )

! ! UNAIRE ! ! fact_non (141 )  :
   *   *   *   *   *   *   *   fact   (139 )

! ! UNAIRE ! ! texp (142 )      :
   *   *   *   *   *   *   *   expr   (148 )

! ! ! SC ! ! ! para_effe (143 ) :
   *   *   *   *   *   *   *   expr   (148 )

! ! ! SC ! ! ! ense (144 )      :
   *   *   *   *   *   *   *   inte_01   (145 )
   *   *   *   *   *   *   *   inte_02   (154 )

! ! UNAIRE ! ! inte_01 (145 )   :
   *   *   *   *   *   *   *   list_elem   (146 )

! ! ! SC ! ! ! list_elem (146 ) :
   *   *   *   *   *   *   *   list_elem_1   (18 )

! ! ! SC ! ! ! elem (147 )      :
   *   *   *   *   *   *   *   vari_indi   (67 )
   *   *   *   *   *   *   *   desi_cham   (68 )
   *   *   *   *   *   *   *   vari_pointee_ou_tamp_fich   (138 )
```

```
! ! ! SC ! ! ! expr (148 )        :
   *   *   *   *   *   *   *    expr_simple   (149 )
   *   *   *   *   *   *   *    test_expr   (29 )

! ! ! SC ! ! ! expr_simple (149 )        :
   *   *   *   *   *   *   *    expr_oper   (30 )
   *   *   *   *   *   *   *    term   (153 )
   *   *   *   *   *   *   *    term_sign   (70 )

! ! ! SC ! ! ! oper_1 (150 )     :
   *   *   *   *   *   *   *    oper_11   (155 )
   *   *   *   *   *   *   *    oper_12   (156 )
   *   *   *   *   *   *   *    oper_13   (157 )
   *   *   *   *   *   *   *    oper_14   (158 )
   *   *   *   *   *   *   *    oper_15   (159 )
   *   *   *   *   *   *   *    oper_16   (170 )
   *   *   *   *   *   *   *    oper_17   (171 )

! ! ! SC ! ! ! oper_2 (151 )     :
   *   *   *   *   *   *   *    oper_21   (172 )
   *   *   *   *   *   *   *    oper_22   (173 )
   *   *   *   *   *   *   *    oper_23   (174 )

! ! ! SC ! ! ! oper_3 (152 )     :
   *   *   *   *   *   *   *    oper_21   (172 )
   *   *   *   *   *   *   *    oper_22   (173 )

! ! ! SC ! ! ! term (153 )        :
   *   *   *   *   *   *   *    fact   (139 )
   *   *   *   *   *   *   *    test_term   (31 )

! ! ! SC ! ! ! oper_4 (154 )     :
   *   *   *   *   *   *   *    oper_41   (175 )
   *   *   *   *   *   *   *    oper_42   (176 )
   *   *   *   *   *   *   *    oper_43   (177 )
   *   *   *   *   *   *   *    oper_44   (178 )
   *   *   *   *   *   *   *    oper_45   (179 )

! ! ! SC ! ! ! const (155 )     :
   *   *   *   *   *   *   *    etiq_sign   (161 )
   *   *   *   *   *   *   *    reel_sign   (158 )
   *   *   *   *   *   *   *    etiq   (181 )
   *   *   *   *   *   *   *    reel   (159 )
   *   *   *   *   *   *   *    chaine   (156 )

! ! UNAIRE ! ! chaine (156 )     :
   *   *   *   *   *   *   *    cara   (157 )
```

```
! ! ! SC ! ! ! cara (157 )       :
  *   *   *   *   *   *   *   etiq   (181 )
  *   *   *   *   *   *   *   ident   (185 )

! ! ! SC ! ! ! reel_sign (158 ) :
  *   *   *   *   *   *   *   reel_1   (71 )
  *   *   *   *   *   *   *   reel_2   (32 )
  *   *   *   *   *   *   *   reel_3   (73 )

! ! ! SC ! ! ! reel (159 )       :
  *   *   *   *   *   *   *   reel_4   (72 )
  *   *   *   *   *   *   *   reel_5   (33 )
  *   *   *   *   *   *   *   reel_6   (74 )

! ! ! SC ! ! ! fact_echel (160 )         :
  *   *   *   *   *   *   *   etiq   (181 )
  *   *   *   *   *   *   *   etiq_sign   (161 )

! ! ! SC ! ! ! etiq_sign (161 ) :
  *   *   *   *   *   *   *   sign_etiq_1   (162 )
  *   *   *   *   *   *   *   sign_etiq_2   (163 )

! ! UNAIRE ! ! sign_etiq_1 (162 )       :
  *   *   *   *   *   *   *   etiq   (181 )

! ! UNAIRE ! ! sign_etiq_2 (163 )       :
  *   *   *   *   *   *   *   etiq   (181 )

! ! ZEROAIRE ! ! int6_02 (164 ) :

! ! ZEROAIRE ! ! oper_11 (165 ) :

! ! ZEROAIRE ! ! oper_12 (166 ) :

! ! ZEROAIRE ! ! oper_13 (167 ) :

! ! ZEROAIRE ! ! oper_14 (168 ) :

! ! ZEROAIRE ! ! oper_15 (169 ) :

! ! ZEROAIRE ! ! oper_16 (170 ) :
```

```
! ! ZEROAIRE ! ! oper_17 (171 ) :

! ! ZEROAIRE ! ! oper_21 (172 ) :

! ! ZEROAIRE ! ! oper_22 (173 ) :

! ! ZEROAIRE ! ! oper_23 (174 ) :

! ! ZEROAIRE ! ! oper_41 (175 ) :

! ! ZEROAIRE ! ! oper_42 (176 ) :

! ! ZEROAIRE ! ! oper_43 (177 ) :

! ! ZEROAIRE ! ! oper_44 (178 ) :

! ! ZEROAIRE ! ! oper_45 (179 ) :

! ! ZEROAIRE ! ! vide (180 )    :
```

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  +                                                                              +
  +                              FICHIER GCLOI                                   +
  +                                                                              +
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/*
 * Ce fichier contient les informations relatives a chaque noeud du
 * formalisme.
 *
 * Chaque ligne reprend trois informations:
 *  - la premiere colonne donne le numero de code du noeud
 *  - la deuxieme et la troisieme colonnes donnent le genre du noeud
 *      rencontre. On en trouve de cinq type:
 *                . les noeuds non listes ayant un operateur
 *                . les noeuds non listes ayant deux operateurs
 *                . les noeuds listes
 *                . le noeud goto...
 *                . les identificateurs
 *  - la quatrieme colonne correspond a un numero d'operateur, ceux-ci
 *      se trouvant dans une table du programme principal.
 *
 * Il faut noter que l'acces a cette table est realise par un SEEK et
 * que seules les trois dernieres colonnes sont utilisees, la premiere
 * servant a reperer le code du noeud a la construction.
 * Pour etre utilise, cette table doit commencer au debut du fichier,
 * ce qui implique que cette table doit etre copiee dans un deuxieme
 * fichier et les commentaires retires
 *                                                                          */
```

```
001 0 0 00
002 0 0 00
003 0 0 00
004 0 0 00
005 0 0 00
006 0 0 00
007 0 0 00
008 0 0 00
009 0 0 00
010 0 0 00
011 3 0 02
012 3 0 02
013 3 0 08
014 3 0 14
015 3 0 14
016 3 0 14
017 3 0 14
018 3 0 14
019 0 0 00
020 0 0 00
021 0 0 00
022 0 0 00
023 0 0 00
024 0 0 00
025 0 0 00
026 0 0 00
027 1 0 05
028 2 0 10
029 0 0 00
030 0 0 00
031 0 0 00
032 5 1 00
033 5 1 00
034 0 0 00
035 0 0 00
036 0 0 00
037 0 0 00
038 0 0 00
039 0 0 00
040 0 0 00
041 0 0 00
042 0 0 00
043 0 0 00
044 0 0 00
045 0 0 00
046 0 0 00
047 0 0 00
048 0 0 00
049 0 0 00
050 0 0 00
051 0 0 00
052 0 0 00
053 0 0 00
054 0 0 00
055 0 0 00
```

```
056 0 0 00
057 0 0 00
058 0 0 00
059 0 0 00
060 1 0 04
061 1 0 06
062 1 0 07
063 1 0 03
064 1 0 08
065 1 0 09
066 1 0 13
067 1 0 15
068 1 0 16
069 0 0 00
070 0 0 00
071 5 1 00
072 5 1 00
073 5 1 00
074 5 1 00
075 0 0 00
076 0 0 00
077 0 0 00
078 0 0 00
079 0 0 00
080 0 0 00
081 0 0 00
082 0 0 00
083 0 0 00
084 0 0 00
085 0 0 00
086 0 0 00
087 0 0 00
088 0 0 00
089 0 0 00
090 0 0 00
091 0 0 00
092 0 0 00
093 0 0 00
094 0 0 00
095 0 0 00
096 0 0 00
097 0 0 00
098 0 0 00
099 0 0 00
100 0 0 00
101 0 0 00
102 0 0 00
103 0 0 00
104 0 0 00
105 0 0 00
106 0 0 00
107 0 0 00
108 0 0 00
109 0 0 00
110 0 0 00
111 0 0 00
```

```
112 0 0 00
113 0 0 00
114 0 0 00
115 0 0 00
116 0 0 00
117 0 0 00
118 0 0 00
119 0 0 00
120 1 0 01
121 0 0 00
122 0 0 00
123 0 0 00
124 0 0 00
125 0 0 00
126 0 0 00
127 1 5 00
128 0 0 00
129 0 0 00
130 0 0 00
131 0 0 00
132 0 0 00
133 0 0 00
134 1 0 11
135 1 0 12
136 0 0 00
137 0 0 00
138 1 0 17
139 0 0 00
140 0 0 00
141 1 0 18
142 1 0 19
143 0 0 00
144 0 0 00
145 1 0 15
146 1 0 15
147 0 0 00
148 0 0 00
149 0 0 00
150 0 0 00
151 0 0 00
152 0 0 00
153 0 0 00
154 0 0 00
155 0 0 00
156 5 1 00
157 0 0 00
158 0 0 00
159 0 0 00
160 0 0 00
161 0 0 00
162 5 1 00
163 5 1 00
164 0 0 00
165 1 0 20
166 1 0 21
167 1 0 22
```

```
168 1 0 23
169 1 0 24
170 1 0 25
171 1 0 34
172 1 0 26
173 1 0 27
174 1 0 28
175 1 0 29
176 1 0 30
177 1 0 31
178 1 0 32
179 1 0 33
180 0 0 00
181 0 1 00
182 0 0 00
183 0 0 00
184 0 0 00
185 0 1 00
```

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  +                                                                            +
  +                          FICHIER DEF.C                                     +
  +                                                                            +
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/*
 * Ce fichier contient les initialisations des codes correspondant
 * a certain noeud utilise dans le programme Fmem.c. Ce fichier sert
 * donc essentiellement a la definition de constantes internes, les
 * valeurs de celles-ci etant donnees par le transformateur.
 *                                                                          */
```

```
# define CIFTHENELSE      27   /* noeud <if_then_else>   */
# define CINSTPOUR        28   /* noeud <inst_pour>      */
# define CREEL2           32   /* noeud <reel_2>         */
# define CREEL5           33   /* noeud <reel_5>         */
# define CINT501          59   /* noeud <int5_01>        */
# define CIFTHEN          61   /* noeud <if_then>        */
# define CINSTCAS         62   /* noeud <inst_cas>       */
# define CINSTTANTQUE     64   /* noeud <inst_tant_que>  */
# define CINSTREPE        65   /* noeud <inst_repe>      */
# define CRECAPPEL        69   /* noeud <rec_appel>      */
# define CREEL1           71   /* noeud <reel_1>         */
# define CREEL4           72   /* noeud <reel_4>         */
# define CREEL3           73   /* noeud <reel_3>         */
# define CREEL6           74   /* noeud <reel_6>         */
# define CPARTINST       120   /* noeud <part_inst>      */
# define CINSTAPPE       126   /* noeud <inst_appe>      */
# define CGOTO           127   /* noeud <inst_bran>      */
# define CCHAINE         156   /* noeud <chaine>         */
# define CSIGNETIQ1      162   /* noeud <sign_etiq_1>    */
# define CSIGNETIQ2      163   /* noeud <sign_etiq_2>    */
# define CETIQ          -131   /* noeud <etiq>           */
# define CIDENT         -135   /* noeud <ident>          */
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  +                                                                             +
  +                              FICHIER MAKEFILE                               +
  +                                                                             +
  +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/* Ce fichier contient les commandes necessaires a la compilation
 * du programme principal.
 *                                                                             */
```

```
Jdata = /users/students/greg/Transfo/data
Jlib  = ${Jdata}/Jans.lib
Fn    = /users/students/greg/Transfo/h

GOAL : .Fmem

.Fmem : Fmem.c $(Jlib)
        cc -g -o Fmem Fmem.c -I$(Fn) $(Jlib) -lm
        touch .Fmem
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  +                                                                           +
  +                          FICHIER  Fmembis.c                               +
  +                                                                           +
  +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/*
 * Ce programme realise la mesure automatique des coefficients de complexite
 * de Halstead et McCabe.Pour se faire,il utilise un analyseur syntaxique
 * servant a la representation interne du programme a mesurer,un outil de
 * parcours de la representation interne permettant de calculer
 * le nombre d'operateurs et d'operandes,un outil de creation d'arbre
 * servant a la memorisation des differents operateurs et operandes,
 * un outil de parcours d'arbre binaire
 * et enfin un outil de calcul des coefficients de complexite.
 *                                                                          */
```

```
/*
 *  Fichier a inclure
 *  -----------------
 *                                                              */

# include "FOGsimu.str" /* Fichier necessaire a l'analyseur syntaxique */
# include "stdio.h"     /* Fichier necessaire aux I/O                  */
# include "math.h"      /* Fichier necessaire a la realisation de log  */
# include "def.c"       /* Fichier contenant les codes de certains noeuds
                           obtenu apres transformation du formalisme   */


/*
 *  Definition des constantes
 *  -------------------------
 *                                                              */

# define  MAX          11   /* grandeur du buffeur buf necessaire
                               a la lecture du fichier argv[2]      */
# define  FTNC       +001   /* grandeur des strings dans memter     */
# define  TMAX         50   /* grandeur de strings internes         */
# define  T2MAX        30   /* grandeur de strings internes         */


/*
 *  Declaration des entiers
 *  -----------------------
 *
 *   FOGdep       : utilise par l'analyseur syntaxique
 *   gli          : utilise par l'analyseur syntaxique
 *   operateur    : donne le nombre total d'operateurs
 *   operande     : donne le nombre total d'operandes
 *   fichnno      : pointeur vers le fichier contenant les separateurs
 *   nbre         : n1 + n2 ( voir Halstead )
 *   nbretotal    : N1 + N2 ( voir Halstead )
 *   vg1          : represente la complexite definie par McCabe
 *   vg2          : represente la complexite Cycmin
 *                                                              */

int FOGdep,operateur,operande,fichnno,gli,nbre,nbretotal,vg1,vg2 ;


/*
 *  Declaration des pointeurs
 *  -------------------------
 *
 *   sommet    : structure de type noeud sommet de l'arbre syntaxique
 *   ref       : structure de type noeud servant de reference
 *   racop     : sommet de l'arbre contenant les operateurs
 *   racoper   : sommet de l'arbre contenant les operandes
 *                                                              */

struct noeud *sommet,*ref ;
```

```
struct ptnoeud *racop,*racoper;

/*
 *  Directory data
 *  --------------
 *
 *   Directory data necessaire a l'utilisation de l'environnement
 *   syntaxique.
 *                                                                  */

char *Adatair;
FILE *Fmsgerr;


/*
 *  Declaration des structures
 *  --------------------------
 *
 *   memter   : structure permettant la memorisation des generiques
 *     - prece : pointeur vers l'element precedent de type memter
 *     - suiva : pointeur vers l'element suivant de type memter
 *     - cnchar: string de longueur FTNC contenant les generiques
 *               bout a bout
 *
 *   noeud    : structure permettant la representation de l'arbre
 *         syntaxique
 *     pour les noeuds non generiques:
 *     - pere  : pointeur vers l'element pere de type noeud
 *     - fils  : pointeur vers le premier element fils de type noeud
 *     - frere : pointeur vers le premier element frere de type noeud
 *     - code  : indique le type de noeud rencontre
 *     - val   : non utilise
 *     pour les noeuds generiques:
 *     - pere  : pointeur vers l'element pere de type noeud
 *     - frere : pointeur vers le premier frere droit de type noeud
 *     - code  : oppose du code du generique
 *     - fils  : reference a un objet de type noeud dont les
 *                caracteristiques sont :
 *               . fils : inutilise
 *               . code : longeur du generique
 *               . pere : adresse de memorisation du premier caractere
 *                        du generique
 *               . frere: adresse de la structure de type memter dans
 *                        laquelle le premier caractere est memorise
 *
 *   ptnoeud  : structure permettant la memorisation de valeurs
 *        particuliere dans un arbre binaire
 *     - gauche : pointeur vers l'element gauche de type ptnoeud
 *     - droite : pointeur vers l'element droite de type ptnoeud
 *     - valnoeud : la valeur de l'element
 *     - count : le nombre de fois que l'element est rencontre
 *                                                                  */

struct memter
 {
  struct memter *prece;
```

```
  struct memter *suiva;
  char cncar[FINC];
 };

struct noeud
 {
  struct noeud *pere;
  struct noeud *frere;
  struct noeud *fils;
  int code;
  int val;
 };

struct ptnoeud
 {
  struct ptnoeud *gauche,*droite;
  char valnoeud[T2MAX];
  int count;
 }
```

```
/*
 *  PROGRAMME PRINCIPAL
 *  -------------------
 *                                                              */

main (argc,argv)

int argc;
char *argv[];

/*
 * Les variables :
 * ---------------
 * externes :
 * - argc           : le nombre d'arguments dans la ligne de commande
 * - argv           : un pointeur vers un vecteur de strings contenant
 *                        ces arguments
 * internes :
 * - fichin         : un pointeur vers un fichier texte a traiter
 * - rac            : pointeur vers une structure de type noeud
 * - start          : pointeur vers une stucture de type FDuples
 * - n1             : nombre d'operateurs distincts
 * - N1             : nombre total d'operateurs
 * - n2             : nombre d'operandes distinctes
 * - N2             : nombre total d'operandes
 * - v              : volume du programme definit selon Halstead
 * - l              : niveau du programme definit selon Halstead
 * - e              : effort de programmation ou coefficient de complexite
 * - nbreaux        : valeur intermediaire
 * - nbreaux2       : valeur intermediaire
 * - nbredou        : valeur intermediaire
 * - N              : longeur du programme
 *
 * Les fonctions :
 * ---------------
 * - ansynt         : fonction necessaire a l'analyseur syntaxique
 * - FDgtst         : fonction necessaire a l'analyseur syntaxique
 * - logarith2      : calcul du logarithme en base 2
 *
 * L'utilisation :
 * ---------------
 * L'utilisateur indiquera "Fmem" suivi par deux noms de fichiers:
 *  - le nom du fichier contenat le programme a analyser
 *  - le nom du fichier contenant les codes relatifs aux operateurs
 *                                                              */


{
 FILE *fichin ;
 struct noeud *ansynt (),*rac ;
 struct FDuples *start,*FDgtst() ;
 int n1,N1,n2,N2 ;
 double v,l,e,nbreaux,nbreaux2,nbredou,N,logarith2() ;

 printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
```

```
 printf("**********************************************************************\n");
 printf("**********************************************************************\n");
 printf("**                                                                  **\n");
 printf("**                                                                  **\n");
 printf("**         MeSURE DE COMPLEXITE DE PROGRAMMES                       **\n");
 printf("**             ECRITS EN PASCAL                                     **\n");
 printf("**                                                                  **\n");
 printf("**                                                                  **\n");
 printf("**          memoire realise par GREGOIRE CH.                        **\n");
 printf("**                                                                  **\n");
 printf("**                        95-96                                     **\n");
 printf("**                                                                  **\n");
 printf("**********************************************************************\n");
 printf("**********************************************************************\n");
 printf("\n\n\n");

 if ( argc == 0)
     {
      printf(" Vous oubliez vos noms de fichiers !!! \n");
      exit();
     }

/*
 * Ouverture des fichiers et controle de presence de ceux-ci
 *                                                                    */

fichin = fopen( argv[1], "r" ) ;
if ( fichin == NULL )
    {
    fprintf( stderr, " Fichier %s non present !!! \n",argv[1]) ;
    exit() ;
    }

fichnno = open( argv[2], 0 ) ;
if ( fichnno == -1 )
    {
    fprintf( stderr, " Fichier %s non present !!! \n",argv[2] ) ;
    exit() ;
    }

/*
 * Initialisation
 *                                                                    */

Adatdir = "/users/students/greg/Transfo/data/";
Fmsgerr = stdout;
gli = 0 ;
FDGdep = 10 ;
operateur = 0 ;
operande = 0 ;
rac = NULL ;
racop = NULL ;
racoper = NULL ;
vgl = 0 ;

/*
```

```
 * Analyse syntaxique : construction de l'arbre correspondant au
 * programme introduit dans fichin.
 *                                                                */

sommet = ansynt( fichin ) ;
if ( sommet == NULL )
   {
     printf( " Pas de reference a un sommet !!! \n\n" ) ;
     exit() ;
   }

/*
 * Decompilation de l'arbre
 *                                                                */

printf ( "      ************************\n");
printf ( "      * PROGRAMME A MESURER *\n");
printf ( "      ************************\n\n");
FDuflI( sommet, 30, stdout ) ;
printf("\n\n") ;

/*
 * Visualisation de l'arbre
 *                                                              */

printf ( "      ********************************\n");
printf ( "      *  VISUALISATION DE L'ARBRE  *\n");
printf ( "      ********************************\n\n");
FDarbvisu( sommet ) ;
printf("\n\n") ;

/*
 * Calcul du nombre d'operandes et d'operateurs
 *                                                                */

rac = sommet->fils->frere;
position( rac ) ;

/*
 * Fermeture des fichiers
 *                                                                */

close( fichnno ) ;
fclose( fichin ) ;

/*
 * Impression des resultats
 *                                                                */

printf("      ********************\n");
printf("      *  LES OPERATEURS  *\n");
printf("      ********************\n\n");
printf(" Nombre total d'operateurs  : %3d\n\n",operateur);
printf(" Les operateurs : \n");
inorder(racop);
n1 = nbre;
```

```
N1 = nbretotal;
printf("\n\n Nombre d'operateurs distincts : %3d\n\n",n1);
nbre = 0;
nbretotal = 0;
printf("       **********************\n");
printf("       *  LES OPERANDES  *\n");
printf("       **********************\n\n");
printf(" Nombre total d'operandes  : %3d\n\n",operande);
printf(" Les operandes : \n");
inorder(racoper);
n2 = nbre;
N2 = nbretotal;
printf("\n\n Nombre d'operandes distincts : %3d\n\n",n2);

/*
 *  Calcul des complexites
 *                                                                    */

printf ("       **********************\n");
printf ("       *  LES COMPLEXITES  *\n");
printf ("       **********************\n\n");
vg1 = vg1 + 1;
printf(" La coefficient de McCabe : %3d\n\n",vg1);
vg2 = vg2 + 1;
printf(" L'intervalle Cycmin:N1 : %3d : %3d\n\n",vg2,operateur);
nbretotal = operateur + operande;
nbre = n1 + n2 ;
nbredou = n1;
nbreaux = logarith2(nbredou,2.);
N = n1 * nbreaux;
nbredou = n2;
nbreaux = logarith2(nbredou,2.);
N = N + n2 * nbreaux;
printf(" Longueur du programme : %13e\n\n",N);
nbredou = nbre;
nbreaux = logarith2(nbredou,2.);
v = nbretotal * nbreaux;
printf(" Volume du programme : %13e\n\n",v);
nbreaux = 2 * n2;
nbreaux2 = n1 * operande;
l = nbreaux / nbreaux2;
printf(" Niveau du programme : %13e\n\n",l);
e = v / l;
printf(" Le coefficient de Halstead : %13e\n\n",e );
}
```

```
/*-------------------------------------------------------------------------*/

double logarithm2(nb1,nb2)
/*=======================*/

/*-------------------------------------------------------------------------*/

/*
 * Procedure qui calcule le logarithme en base 2
 *
 * variables:
 *   input: nb1,nb2 :les nombres a traiter
 *   interne: interm,interm2 : variables intermediaires
 *                                                                         */

double nb1,nb2;

{
  double interm,interm2;

  interm = log(nb1);
  interm2 = log(nb2);
  interm = interm/interm2;
  return(interm);
}
```

```
/*-----------------------------------------------------------------------*/

position(pr)
/*==========*/

/*-----------------------------------------------------------------------*/

/*
 * Procedure de parcours de l'arbre syntaxique
 *
 * variables:
 *   i/o   : pr :pointeur vers un noeud de l'arbre a traiter
 *                                                                       */

struct noeud *pr;

{
 if (pr != NULL)
  {
   if (pr->code != CPARTINST)
    {
     if ((pr->code != CIDENT) && (pr->code != CETIQ)) position(pr->fils);
     position(pr->frere);
    }
   else {
         lirefils(pr);
        }
  }
}
```

```
/*-----------------------------------------------------------------------*/

lirefils(pr)
/*=========*/

/*-----------------------------------------------------------------------*/

/*
 * Procedure de parcours de la partie instruction de l'arbre syntaxique
 * et calcul du nombre d'operandes et d'operateurs.
 *
 * Variables:
 *   i/o    : pr :noeud de l'arbre a traiter
 * interne  : buf :variable qui va contenir la string lue par la fonction
 *                 get
 *            mot :string auxiliaire contenant les valeurs des operandes
 *                 ou des operateurs
 *            valeur :entier contenant le nombre de fils d'un noeud dans
 *                    certain cas
 *            ent1 :contient le code correspondant a l'operateur
 *            ent2 :contient le code correspondant a l'operande
 *            ent3 :contient la reference a un operateur
 *            sortie :indique le noeud suivant a lire
 *              = 1 :on passe au noeud frere
 *              = 0 :on passe au noeud fils
 *            fin :longueur de strings sortant de Mvltxt
 *
 * Fonctions:
 *   nbrefils     :donne le nombre de fils d'un noeud
 *   arbre        :procedure de parcours de l'arbre
 *   Mvltxt       :procedure donnant la valeur du generique
 *                                                                      */

struct noeud *pr;

{
 char buf[MAX],mot[T2MAX];
 int valeur,ent1,ent2,ent3,nbrefils(),sortie,fin,Mvltxt();
 struct ptnoeud *arbre();

  if (pr != NULL)
   {

    if (pr->code >0)  ent1 = pr->code - 1;
    else ent1 = 0 - pr->code -1;

    /*
     *  Lecture des codes sur le fichier fichnno
     *                                                                  */

    get( fichnno,ent1*11,buf,11 ) ;
    ent1 = buf[4]-'0' ;
    ent2 = buf[5]-'0' ;
    ent3 = (buf[8]-'0')*10 + (buf[9]-'0');
    sortie = 0;
```

```
    mccabe(pr);
    cycmin(pr);

    /*
     * le noeud n'est pas une liste et contient un seul operateur
     *                                                          */

    if ((ent1 == 1) && (ent2 != 5))
     {
      operateur++;
      correspoper(ent3,mot);
      racop = arbre(mot,1,racop);
      sortie = 0;
     }

    /*
     * le noeud n'est pas une liste mais contient deux operateurs
     *                                                          */

    if ((ent1 == 2) && (ent2 != 5))
     {
      operateur++;
      correspoper(ent3,mot);
      racop = arbre(mot,1,racop);
      ent3 = 4;
      correspoper(ent3,mot) ;
      racop = arbre(mot,1,racop);
      sortie = 0;
     }

    /*
     * le noeud est une liste
     *                                                          */

    if ((ent1 == 3) && (ent2 != 5))
     {
      valeur = norefils(pr);
      sortie = 0;
      if ( valeur > 1 )
       {
        operateur = operateur+valeur-1;
        correspoper(ent3,mot);
        racop = arbre(mot,valeur-1,racop);
       }
     }

    /*
     * le noeud n'est pas une liste mais une instruction de type goto...
     *                                                          */

    if ((ent1 == 1) && (ent2 == 5))
     {
      operateur++;
      sortie = 1;
      pr = pr->fils;
      mot[0]='g';
```

```
        mot[1]='o';
        mot[2]='t';
        mot[3]='o';
        mot[4]=' ';
        fin = Mvltxt(pr,mot,3);
        mot[fin]='\0';
        racop = arbre(mot,1,racop);
       }

      /*
       * le noeud est un identificateur de fonction
       *                                                          */

      if ((ent2 == 1) && (ent1 != 5) &&
        ((pr->pere->code == CINSTAPPE) || (pr->pere->code == CRECAPPEL)))
       {
        operateur++;
        sortie = 1;
        fin = Mvltxt(pr,mot,0);
        mot[fin] = '\0';
        racop = arbre(mot,1,racop);
       }

      /*
       * le noeud est un identificateur
       *                                                          */

      if ((ent2 == 1) && (ent1 != 5) && (pr->pere->code != CINSTAPPE)
          && (pr->pere->code != CRECAPPEL) && (pr->pere->code != CINT501))
       {
        operande++;
        sortie = 1;
        fin = Mvltxt(pr,mot,0);
        mot[fin] = '\0';
        racoper = arbre(mot,1,racoper);
          }

      /*
       * le noeud est un label
       *                                                          */

      if ((ent2 == 1) && (ent1 !=5) && (pr->pere->code == CINT501))
       {
        sortie = 1;
       }

      /*
       * le noeud est un identificateur necessitant une concatenation
       *                                                          */

      if ((ent2 == 1) && (ent1 == 5))
       {
        operande++;
        sortie = 1;
        calcul(pr,mot);
        racoper = arbre(mot,1,racoper);
```

```
        }

       /*
        * lecture des noeuds suivants
        *                                                          */

       if (sortie != 1) lirefils( pr->fils );
       lirefils( pr->frere ) ;
     }
}
```

```
/*-----------------------------------------------------------------------*/

mccabe(pr)
/*=======*/

/*-----------------------------------------------------------------------*/

/*
 * procedure qui determine si le noeud est un point de decision ou non
 *
 * Variables:
 *   input : pr :pointeur vers un noeud a traiter
 *  interne: int :valeur intermediaire
 *                                                                       */
struct noeud *pr;

{
 int valeur;

 if ((pr->code == CIFTHEN) || (pr->code == CIFTHENELSE) ||
     (pr->code == CINSTTANTQUE) || (pr->code == CINSTREPE) ||
        (pr->code == CINSTPOUR))
    vgl = vgl + 1;
 if (pr->code == CINSTCAS)
  {
   valeur = nbrefils(pr->fils->frere);
   vgl = vgl + valeur-1;
  }
}
```

```
/*-----------------------------------------------------------------------*/

cycmain(pr)
/*=======*/

/*-----------------------------------------------------------------------*/

/*
 * procedure qui determine si le noeud est un point de decision ou non
 *
 * Variables:
 *   input : pr :pointeur vers un noeud a traiter
 *                                                                       */

struct noeud *pr;

{
 if ((pr->code == CIFTHEN) || (pr->code == CIFTHENELSE) ||
     (pr->code == CINSTTANTQUE) || (pr->code == CINSTREPE) ||
        (pr->code == CINSTPOUR) || (pr->code == CINSTCAS))
   vg2 = vg2 + 1;
}
```

```
/*-----------------------------------------------------------------------*/

get(fd,pos,buf,n)
/*==============*/

/*-----------------------------------------------------------------------*/

/*
 * Procedure qui lit dans un fichier fd a partir de la position pos
 * une string de longeur n et la place dans buf
 *
 *  Variables:
 *    input : fd :nom du fichier ou doit se pratiquer la lecture
 *            pos :la position a partir de laquelle il faut lire
 *            n :le nombre de bytes qu'il faut lire
 *    output :buf :contient la string de longueur n lue
 *                                                                    */

int fd,n ;
long pos ;
char *buf ;

{
  lseek( fd,pos,0 ) ;
  return( read( fd,buf,n ) ) ;
}
```

```
/*-------------------------------------------------------------------------*/

inorder(rac)
/*=========*/

/*-------------------------------------------------------------------------*/

/*
 * procedure de lecture d'un arbre binaire
 *
 * Variable:
 *   i/o  : rac :pointeur vers le noeud a traiter
 *                                                                         */

struct ptnoeud *rac;

{
  if ( rac != NULL)
  {
   inorder(rac->gauche);
   printf("        %-30s %3d\n",rac->valnoeud,rac->count);
   nbre = nbre + 1;
   nbretotal = nbretotal + rac->count;
   inorder(rac->droite);
  }
}
```

```
/*-----------------------------------------------------------------------*/

struct ptnoeud *arbre(mot,somme,rac)
/*=======================================*/

/*-----------------------------------------------------------------------*/

/*
 * procedure d'introduction d'un noeud dans un arbre binaire
 *
 * Variables:
 *  input : mot :la valeur du noeud a introduire dans l'arbre
 *          somme :le poids de ce noeud
 *  i/o   : rac :pointeur vers le noeud a traiter
 * interne: cond :indique la comparaison entre mot et la valeur du
 *                noeud traite
 *
 * Fonctions:
 *  talloc :procedure d'allocation d'espace de stockage du noeud
 *  arbre  :procedure d'introduction d'un noeud dans l'arbre
 *                                                              */

struct ptnoeud *rac;
char mot[T2MAX];
int somme;

{
 struct ptnoeud *talloc(),*arbre();
 int cond;

 if ( rac == NULL )
  {
   rac = talloc() ;
   strcpy(rac->valnoeud,mot) ;
   rac->count = somme ;
   rac->gauche = NULL;
   rac->droite = NULL;
  }
 else if (( cond = strcmp(mot,rac->valnoeud)) == 0)
       rac->count = rac->count + somme;
      else if ( cond < 0 )
            rac->gauche = arbre ( mot,somme,rac->gauche );
           else
            rac->droite = arbre ( mot,somme,rac->droite );
 return(rac);
}
```

```
/*-----------------------------------------------------------------------*/

struct ptnoeud *talloc()
/*====================*/

/*-----------------------------------------------------------------------*/

/*
 * procedure d'allocation de place pour le sauvetage d'un noeud
 *
 * Fonction:
 *  malloc :allocation de l'espace de sauvetage
 *                                                                       */

{
 char *malloc();

 return((struct ptnoeud *) malloc (sizeof( struct ptnoeud )));

}
```

```
/*-------------------------------------------------------------------------*/

int nbrefils(pr)
/*==========*/

/*-------------------------------------------------------------------------*/

/*
 * procedure de calcul du nombre de fils d'un noeud
 *
 * Variables:
 *  input : pr :le noeud pour lequel il faut chercher le nombre de fils
 * interne: prav :un pointeur intermediaire
 *          i :le nombre de fils rencontre
 *                                                                         */

struct noeud *pr;

{
 struct noeud *prav;
 int i;

 prav = pr->fils;
 i = 1;
 while ( prav->frere != NULL )
  {
   i++;
   prav = prav->frere;
  }
 return(i);
}
```

```
/*-------------------------------------------------------------------------*/

corresoper(numero,mot2)
/*=======================*/

/*-------------------------------------------------------------------------*/

/*
 * procedure etablissant la correspondance entre un code et l'operateur
 * correspondant
 *
 * Variables:
 *    input : numero :le code designant l'operateur
 *   output : mot2 :l'operateur correspondant
 *                                                                         */

int numero;
char mot2[TAILLEMAX];

{
 char *mot;

 if ( numero == 1 ) mot = "begin";
 if ( numero == 2 ) mot = ";";
 if ( numero == 3 ) mot = ":";
 if ( numero == 4 ) mot = ":=";
 if ( numero == 5 ) mot = "if+else";
 if ( numero == 6 ) mot = "if";
 if ( numero == 7 ) mot = "case";
 if ( numero == 8 ) mot = "while";
 if ( numero == 9 ) mot = "repeat";
 if ( numero == 10 ) mot = "for";
 if ( numero == 11 ) mot = "to";
 if ( numero == 12 ) mot = "downto";
 if ( numero == 13 ) mot = "with";
 if ( numero == 14 ) mot = ",";
 if ( numero == 15 ) mot = "[]";
 if ( numero == 16 ) mot = ".";
 if ( numero == 17 ) mot = "^";
 if ( numero == 18 ) mot = "not";
 if ( numero == 19 ) mot = "()";
 if ( numero == 20 ) mot = "<>";
 if ( numero == 21 ) mot = ">";
 if ( numero == 22 ) mot = "<";
 if ( numero == 23 ) mot = ">=";
 if ( numero == 24 ) mot = "<=";
 if ( numero == 25 ) mot = "in";
 if ( numero == 26 ) mot = "+";
 if ( numero == 27 ) mot = "-";
 if ( numero == 28 ) mot = "or";
 if ( numero == 29 ) mot = "*";
 if ( numero == 30 ) mot = "/";
 if ( numero == 31 ) mot = "div";
 if ( numero == 32 ) mot = "mod";
 if ( numero == 33 ) mot = "and";
 if ( numero == 34 ) mot = "=";
```

```
 strcpy(mot2,mot);
}
```

```
/*-------------------------------------------------------------------------*/

calcul(pr,str)
/*===========*/

/*-------------------------------------------------------------------------*/

/*
 * procedure de concatenation des genériques
 *
 * Variables:
 *   input : pr :le noeud de depart du generique
 *   output: str:la valeur de la concatenation en sortie
 *  interne: fin,fin2 :entiers intermediaires
 *
 * Fonction:
 *  Mvltxt :procedure donnant la valeur du generique
 *                                                                  */

struct noeud *pr;
char str[TAMAX];

{
 int fin,fin2,Mvltxt();

 str[0] = '\0';

 /*
  * le generique est de la forme 'aaaaaaaaa'
  *                                                                 */

 if ( pr->code == CCHAINE )
  {
   pr = pr->fils;
   str[0] = '\'';
   fin = Mvltxt(pr,str,1);
   str[fin] = '\'';
   str[fin+1] = '\0';
  }

 /*
  * le generique est un entier signe de la forme +...
  *                                                                 */

 if ( pr->code == CSIGNETIQ1 )
  {
   pr = pr->fils;
   str[0] = '+';
   fin = Mvltxt(pr,str,1);
   str[fin] = '\0';
  }

 /*
  * le generique est un entier signe de la forme -...
  *                                                                 */
```

```
 if ( pr->code == CSIGNETIQ2 )
  {
   pr = pr->fils;
   str[0] = '-';
   fin = Mvltxt(pr,str,1);
   str[fin] = '\0';
  }

 /*
  * le generique est un reel non signe sans exposant
  *                                                          */

 if ( pr->code == CREEL4 )
  {
   pr = pr->fils;
   fin = Mvltxt(pr,str,0);
   str[fin] = '.';
   pr = pr->frere;
   fin2 = Mvltxt(pr,str,fin+1);
   str[fin2] = '\0';
  }

 /*
  * le generique est un reel sans exposant
  *                                                          */

 if ( pr->code == CREEL1 )
  {
   pr = pr->fils;
   if (pr->code == CSIGNETIQ1)
     {
       str[0] = '+';
       fin = Mvltxt(pr->fils,str,1);
     }
   else
     {
       str[0] = '-';
       fin = Mvltxt(pr->fils,str,1);
     }
   pr = pr->frere;
   str[fin] = '.';
   fin2 = Mvltxt(pr,str,fin+1);
   str[fin2] = '\0';
  }


 /*
  * le generique est un reel avec exposant
  *                                                          */

 if ( pr->code == CREEL3 )
  {
   pr = pr->fils;
   if (pr->code == CSIGNETIQ1)
     {
       str[0] = '+';
```

```
        fin = Mvltxt(pr->fils,str,1);
      }
    else
      {
        str[0] = '-';
        fin = Mvltxt(pr->fils,str,1);
      }
    str[fin] = 'e';
    pr = pr->frere;
    if (pr->code == CETIQ )
          {
           fin2 = Mvltxt(pr,str,fin+1);
           str[fin2] = '\0';
          }
    else
       {
      if (pr->code == CSIGNETIQ1)
      {
        str[fin+1] = '+';
        fin2 = Mvltxt(pr->fils,str,fin+2);
        str[fin2] = '\0';
       }
    else
      {
        str[fin+1] = '-';
        fin2 = Mvltxt(pr->fils,str,fin+2);
        str[fin2] = '\0';
      }
        }
  }

/*
 * le generique est un entier avec exposant
 *                                                              */

if ( pr->code == CREEL6 )
 {
  pr = pr->fils;
  fin = Mvltxt(pr,str,0);
  str[fin] = 'e';
  pr = pr->frere;
  if (pr->code == CETIQ )
        {
         fin2 = Mvltxt(pr,str,fin+1);
         str[fin2] ='\0';
        }
  else
     if (pr->code == CSIGNETIQ1)
       {
        str[fin+1] = '+';
        fin2 = Mvltxt(pr->fils,str,fin+2);
        str[fin2] = '\0';
       }
     else
       {
        str[fin+1] ='-';
```

```
            fin2 = Mvltxt(pr->fils,str,fin+2);
            str[fin2] = '\0';
           }
  }

 /*
  * le generique est un reel signe avec exposant
  *                                                              */

 if ( pr->code == CREEL2 )
  {
   pr = pr->fils;
   if (pr->code == CSIGNETIQ1)
     {
       str[0] = '+';
       fin = Mvltxt(pr->fils,str,1);
     }
   else
     {
       str[0] = '-';
       fin = Mvltxt(pr->fils,str,1);
     }
   str[fin] = '.';
   pr = pr->frere;
   fin2 = Mvltxt(pr,str,fin+1);
   str[fin2] = 'e';
   pr = pr->frere;
   if (pr->code == CETIQ )
          {
           fin = Mvltxt(pr,str,fin2+1);
           str[fin] = '\0';
          }
   else
      if (pr->code == CSIGNETIQ1)
       {
        str[fin2+1] = '+';
        fin = Mvltxt(pr->fils,str,fin2+2);
        str[fin] = '\0';
       }
     else
       {
        str[fin2+1] = '-';
        fin = Mvltxt(pr->fils,str,fin2+2);
        str[fin] = '\0';
       }
  }

 /*
  * le generique est un reel non signe avec exposant
  *                                                              */

 if ( pr->code == CREEL5 )
  {
   pr = pr->fils;
   fin = Mvltxt(pr,str,0);
   str[fin] = '.';
```

```
    pr = pr->frere;
    fin2 = Mvltxt(pr,str,fin+1);
    str[fin2] = 'e';
    pr = pr->frere;
    if (pr->code == CETIQ )
            {
             fin = Mvltxt(pr,str,fin2+1);
             str[fin] = '\0';
            }
    else
      if (pr->code == CSIGNETIQ1)
        {
          str[fin2+1] ='+';
          fin = Mvltxt(pr->fils,str,fin2+2);
          str[fin] = '\0';
        }
      else
       {
          str[fin2+1] = '-';
          fin = Mvltxt(pr->fils,str,fin2+2);
          str[fin] = '\0';
       }
  }
}
```

```
/*-----------------------------------------------------------------------*/

int Mvltxt (sb_tree,txt,debut)
/*=======================*/

/*-----------------------------------------------------------------------*/

/*
 * procedure recherchant la valeur d'un generique
 *
 * Variables:
 *   input : sb_tree : pointeur vers le noeud generique
 *           debut   : debut du string txt
 *   output: txt :valeur du generique
 *           fin :fin du string txt
 *  interne: lg_gen :longueur du generique
 *           i :pointeur vers le caractere a traiter
 *           str_mem :pointeur vers une structure memter
 *           nod_gen :pointeur vers une structure noeud
 *                                                                    */

struct  noeud   *sb_tree        ;
char    txt[TMAX]                       ;
int debut;

{
        int     lg_gen,j  ;
        char    *i        ;
        struct  memter  *str_mem ;
        struct  noeud   *nod_gen ;


        nod_gen = sb_tree->fils;
        lg_gen  = nod_gen->code;
        str_mem = (struct memter *) nod_gen->frere;
        i       = (char *) nod_gen->pere;
        j       = debut;

        if (lg_gen > TMAX)
                lg_gen = TMAX-1;

        while (lg_gen != 0)
            {
                while (lg_gen != 0 && *i != '\0')
                    {
                        txt[j] = *i;
                        j      += 1 ;
                        lg_gen -= 1 ;
                        i      += 1 ;
                    }
                str_mem = str_mem->suiva;
                i       = &str_mem->chcar[0];
            }
        return(j);
}
```

MESURE DE COMPLEXITE DE PROGRAMMES
ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

35-86

```
        ************************
        * PROGRAMME A MESURER *
        ************************

(* gc *)
program reine ;

type mat8 = array [ 1 .. 8 , 1 .. 8 ] of integer ;
     vect8 = array [ 1 .. 8 ] of integer ;

var a : mat8 ;
    b : vect8 ;
    i , j , k , l , z , m , n : integer ;

procedure nega ( c , d : integer ; var a : mat8 ) ;

begin
  a [ c , d ] := + c ;
  x := c + 1 ;
  while x <= 8 do
    if a [ x , d ] = 0
      then begin
        a [ x , d ] := - c ;
        x := x + 1
      end
      else x := x + 1 ;
  x := 1 ;
  while ( c + x <= 8 ) and ( d + x <= 8 ) do
    begin
      x := c + x ;
      y := d + x ;
      if a [ x , y ] = 0
        then begin
          a [ x , y ] := - c ;
          x := x + 1
        end
        else x := x + 1
    end ;
  x := 1 ;
  while ( c + x <= 8 ) and ( d - x > 0 ) do
    begin
      x := c + x ;
      y := d - x ;
      if a [ x , y ] = 0
        then begin
          a [ x , y ] := - c ;
          x := x + 1
        end
        else x := x + 1
    end
end ;

procedure agen ( c , d : integer ; var a : mat8 ) ;

var x , l : integer ;
```

```
begin
  a [ c , d ] := 0 ;
  for k := c + 1 to 8 do
    for l := 1 to 8 do
      if a [ k , l ] = - c
        then a [ k , l ] := 0
end ;

begin
  i := 1 ;
  j := 1 ;
  z := 1 ;
  b [ 1 ] := 1 ;
  k := 1 ;
  while k <= 8 do
    begin
      i := 1 ;
      while i <= 8 do
        begin
          a [ i , k ] := 0 ;
          i := i + 1
        end ;
      k := k + 1
    end ;
  while z <= 92 do
    if j <= 8
      then begin
        if a [ i , j ] = 0
          then begin
            nega ( i , j , a ) ;
            b [ i ] := j ;
            i := i + 1 ;
            j := 1 ;
            if i = 9
              then begin
                for k := 1 to 8 do
                  begin
                    for l := 1 to 7 do
                      if a [ k , l ] <= 0
                        then write ( 'o' )
                        else write ( 'x' ) ;
                    if a [ k , 8 ] <= 0
                      then writeln ( 'o' )
                      else writeln ( 'x' )
                  end ;
                writeln ( 'b__________________' ) ;
                write ( 'La__' , z ) ;
                writeln ( 'solution_est_' ) ;
                for k := 1 to 7 do
                  write ( b [ k ] ) ;
                writeln ( b [ 8 ] ) ;
                writeln ( 'b__________________' ) ;
                writeln ( 'b__________________' ) ;
                z := z + 1 ;
                i := 7 ;
                a [ 8 , b [ 8 ] ] := 0 ;
```

```
                j := b [ 7 ] + 1 ;
                agen ( 7 , b [ 7 ] , a )
              end
              else j := j + 1
          end
          else begin
            if i < 9
              then begin
                i := i - 1 ;
                j := b [ i ] ;
                agen ( i , j , a ) ;
                j := j + 1
              end
          end
      end
end .
```

```
        ********************************
        *  VISUALISATION DE L'ARBRE    *
        ********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 5  VALUE :reine
! ! ! !  ( int0_01: 20 )
! ! ! ! !  ( int1_06: 37 )
! ! ! ! ! !  ( decl_type: 88 )
! ! ! ! ! ! !  ( suit_decl_type: 89 )
! ! ! ! ! ! ! !  ( suit_type: 3 )
! ! ! ! ! ! ! ! !  ( def_type: 44 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :mat3
! ! ! ! ! ! ! ! ! !  ( type_tabl: 46 )
! ! ! ! ! ! ! ! ! ! !  ( list_indice: 5 )
! ! ! ! ! ! ! ! ! ! ! !  ( type_inte: 45 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :3
! ! ! ! ! ! ! ! ! ! ! !  ( type_inte: 45 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :3
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! !  ( def_type: 44 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 5  VALUE :vect8
! ! ! ! ! ! ! ! ! !  ( type_tabl: 46 )
! ! ! ! ! ! ! ! ! ! !  ( list_indice: 5 )
! ! ! ! ! ! ! ! ! ! ! !  ( type_inte: 45 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :3
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :mat3
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 5  VALUE :vect8
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :l
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :m
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_decl_prfo: 109 )
```

```
! ! ! ! ! !  ( fonct_proc: 10 )
! ! ! ! ! ! !  ( decl_proc_1: 54 )
! ! ! ! ! ! ! !  ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :nega
! ! ! ! ! ! ! ! !  ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! !  ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :mat8
! ! ! ! ! ! ! ! !  ( int0_04: 78 )
! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! !  ( term_sign: 70 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( iden*: -135 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! !  ( inst_tant_que: 64 )
! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( term_sign: 70 )
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_tant_que: 64 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( test_term: 31 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( texp: 142 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_45: 179 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( texp: 142 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( term_sign: 70 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_tant_que: 64 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_term: 31 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( texp: 142 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_45: 179 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( texp: 142 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_12: 166 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( term_sign: 70 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ( decl_proc_1: 54 )
! ! ! ! ! ! ! ! ! ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :agen
! ! ! ! ! ! ! ! ! ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! ! ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :mat8
! ! ! ! ! ! ! ! ( int0_03: 36 )
! ! ! ! ! ! ! ! ! ( int1_15: 83 )
! ! ! ! ! ! ! ! ! ! ( decl_var: 107 )
! ! ! ! ! ! ! ! ! ! ! ( part_decl_var: 108 )
```

```
! ! ! ! ! ! ! ! ! ! ! ! ( decl_var_1: 9 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :l
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ( if_then: 61 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :l
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( term_sign: 70 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :l
! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
```

```
! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! !  ( inst_tant_que: 64 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :3
! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! !  ( inst_tant_que: 64 )
! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :3
! ! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! !  ( inst_tant_que: 64 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :92
! ! ! ! ! ! ! !  ( if_then: 61 )
! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
```

```
! ! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :nega
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :9
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :7
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VAL
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  V
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  V
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 5  VAL
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 5  VAL
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALU
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VAL
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALU
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :(
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALU
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  \
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALU
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  \
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :write
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 19  VALUE :(
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 5  VALUE :write
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :L
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 125 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :write
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 13  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :7
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 5  VALUE :wri
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALU
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :write
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :write
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 19  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 125 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :write
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( chaine: 156 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 19  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :7
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :7
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :agen
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :7
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( if_then: 61 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_13: 167 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :9
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :agen
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
```

```
! ! ! 
```

```
      **********************
      *  LES OPERATEURS    *
      **********************
```

Nombre total d'operateurs : 241

Les operateurs :

()	4
+	22
,	22
-	7
:=	49
;	43
<	1
<=	10
=	6
>	1
[]	23
agen	2
and	2
begin	16
for	5
if	3
if+else	7
nega	1
to	5
wnile	6
write	4
writeln	7

Nombre d'operateurs distincts : 22

```
      **********************
      *  LES OPERANDES     *
      **********************
```

Nombre total d'operandes : 212

Les operandes :

'La__'	1
'b_____'	3
'o'	2
'solution_est_'	1
'x'	2
0	11
1	30
7	6
8	14
9	2
92	1
a	13
b	8
c	13
d	8
i	13
j	13
k	39
l	10
x	6
y	6
z	5

Nombre d'operandes distincts : 22

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 22

L'intervalle Cycmin:Ni : 22 : 241

Longueur du programme : 1.962150e+02

Volume du programme : 2.473123e+03

Niveau du programme : 9.433962e-03

Le coefficient de Halstead : 2.621510e+05

MESURE DE COMPLEXITE DE PROGRAMMES
ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

86-86

```
        **************************
        * PROGRAMME A MESURER *
        **************************

(* gc *)
program reine ;

type mat8 = array [ 1 .. 8 , 1 .. 8 ] of integer ;
     vect8 = array [ 1 .. 8 ] of integer ;

var a : mat8 ;
    b : vect8 ;
    i , j , k , l , z , m , n : integer ;

procedure idem ( c , k , y : integer ; var x : integer ) ;

begin
  x := c + k ;
  if a [ x , y ] = 0
    then a [ x , y ] := - c
end ;

procedure nega ( c , d : integer ; var a : mat8 ) ;

begin
  a [ c , d ] := + c ;
  for k := c + 1 to 8 do
    if a [ k , d ] = 0
      then a [ k , d ] := - c ;
  k := 1 ;
  while ( c + k <= 8 ) and ( d + k <= 8 ) do
    begin
      y := d + k ;
      idem ( c , k , y , x ) ;
      k := k + 1
    end ;
  k := 1 ;
  while ( c + k <= 8 ) and ( d - k > 0 ) do
    begin
      y := d - k ;
      idem ( c , k , y , x ) ;
      k := k + 1
    end
end ;

procedure agen ( c , d : integer ; var a : mat8 ) ;

var k , l : integer ;

begin
  a [ c , d ] := 0 ;
  for k := c + 1 to 8 do
    for l := 1 to 8 do
      if a [ k , l ] = - c
        then a [ k , l ] := 0
end ;
```

```
begin
  i := 1 ;
  j := 1 ;
  z := 1 ;
  b [ 1 ] := 1 ;
  for k := 1 to 8 do
    for l := 1 to 8 do
      a [ l , k ] := 0 ;
  while z <= 92 do
    if j <= 8
      then begin
        if a [ i , j ] = 0
          then begin
            nega ( i , j , a ) ;
            b [ i ] := j ;
            i := i + 1 ;
            j := 1 ;
            if i = 9
              then begin
                for k := 1 to 8 do
                  begin
                    for l := 1 to 7 do
                      if a [ k , l ] <= 0
                        then write ( 'o' )
                        else write ( 'x' ) ;
                    if a [ k , 8 ] <= 0
                      then writeln ( 'o' )
                      else writeln ( 'x' )
                  end ;
                writeln ( 'b__________________' ) ;
                write ( 'La__' , z ) ;
                writeln ( 'solution_est_' ) ;
                for k := 1 to 7 do
                  write ( b [ k ] ) ;
                writeln ( b [ 8 ] ) ;
                writeln ( 'b__________________' ) ;
                writeln ( 'b__________________' ) ;
                z := z + 1 ;
                i := 7 ;
                a [ 8 , b [ 8 ] ] := 0 ;
                j := b [ 7 ] + 1 ;
                agen ( 7 , b [ 7 ] , a )
              end
              else j := j + 1
          end
          else begin
            if i < 9
              then begin
                i := i - 1 ;
                j := b [ i ] ;
                agen ( i , j , a ) ;
                j := j + 1
              end
          end
      end
```

```
end .
```

```
        ********************************
        *  VISUALISATION DE L'ARBRE    *
        ********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 5  VALUE :reine
! ! ! !  ( int0_01: 20 )
! ! ! ! !  ( int1_06: 37 )
! ! ! ! ! !  ( decl_type: 88 )
! ! ! ! ! ! !  ( suit_decl_type: 89 )
! ! ! ! ! ! ! !  ( suit_type: 3 )
! ! ! ! ! ! ! ! !  ( def_type: 44 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :mat8
! ! ! ! ! ! ! ! ! !  ( type_tabl: 46 )
! ! ! ! ! ! ! ! ! ! !  ( list_indice: 5 )
! ! ! ! ! ! ! ! ! ! ! !  ( type_inte: 45 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! !  ( type_inte: 45 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! !  ( def_type: 44 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 5  VALUE :vect8
! ! ! ! ! ! ! ! ! !  ( type_tabl: 46 )
! ! ! ! ! ! ! ! ! ! !  ( list_indice: 5 )
! ! ! ! ! ! ! ! ! ! ! !  ( type_inte: 45 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :mat8
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 5  VALUE :vect8
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :l
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :m
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_decl_prfo: 109 )
```

```
! ! ! ! ! !  ( fonct_proc: 10 )
! ! ! ! ! ! !  ( decl_proc_1: 54 )
! ! ! ! ! ! ! !  ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :idem
! ! ! ! ! ! ! ! !  ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! ! !  ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! !  ( int0_04: 78 )
! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! !  ( if_then: 61 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! !  ( term_sign: 70 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! !  ( decl_proc_1: 54 )
! ! ! ! ! ! ! !  ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :nega
! ! ! ! ! ! ! ! !  ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! ! !  ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :mat3
! ! ! ! ! ! ! ! ! ( int0_04: 78 )
! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ( term_sign: 70 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ( if_then: 61 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( term_sign: 70 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ( inst_tant_que: 64 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( test_term: 31 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( texp: 142 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_45: 179 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( texp: 142 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :idem
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ( inst_tant_que: 64 )
! ! ! ! ! ! ! ! ! ! ! ! ( test_term: 31 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( texp: 142 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_45: 179 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( texp: 142 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_12: 166 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 59 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :idem
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :y
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ( decl_proc_1: 54 )
! ! ! ! ! ! ! ! ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :agen
! ! ! ! ! ! ! ! ! ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! ! ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! ! ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! ! ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! ! ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :mat3
! ! ! ! ! ! ! ! ( int0_03: 36 )
! ! ! ! ! ! ! ! ! ( int1_15: 83 )
! ! ! ! ! ! ! ! ! ! ( decl_var: 107 )
! ! ! ! ! ! ! ! ! ! ! ( part_decl_var: 108 )
! ! ! ! ! ! ! ! ! ! ! ! ( decl_var_1: 9 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :d
! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
```

```
! ! ! ! ! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :l
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ( if_then: 61 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :l
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( term_sign: 70 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :c
! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :l
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ( inst_pour: 28 )
! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ( inst_pour: 28 )
! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
```

```
! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :k
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! !  ( inst_tant_que: 64 )
! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :92
! ! ! ! ! ! ! ! !  ( if_then: 61 )
! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :nega
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_17: 171 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :9
```

```
! ! ! 
```

```
! ! ! 
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :z
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :7
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :8
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :0
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :7
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 4  VALUE :agen
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :7
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( if_then: 61 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( oper_13: 167 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( etiq: -181 )LENGTH : 1  VALUE :9
! ! ! ! ! ! ! ! ! ! ! ! ! ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! ! ! ! ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ( ident: -185 )LENGTH : 1  VALUE :i
```

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( vari_indi: 67 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :b
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_expr: 16 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :agen
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :a
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 216

Les operateurs :

()	4
+	15
,	26
-	6
:=	44
;	36
<	1
<=	7
=	5
>	1
[]	21
agen	2
and	2
begin	12
for	3
idem	2
if	5
if+else	4
nega	1
to	3
while	3
write	4
writeln	7

Nombre d'operateurs distincts : 23

****
* LES OPERANDES *
****

Nombre total d'operandes : 138

Les operandes :

'La__'	1
'b_____'	3
'o'	2
'solution_est_'	1
'x'	2
0	10
1	24
7	6
8	14
9	2
92	1
a	16
b	8
c	13
d	8
i	13
j	13
k	28
l	7
x	6
y	6
z	6

Nombre d'operandes distincts : 22

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 21

L'intervalle Cycmin:Ni : 21 : 216

Longueur du programme : 2.021494e+02

Volume du programme : 2.213709e+03

Niveau du programme : 1.017576e-02

Le coefficient de Halstead : 2.180386e+05

MESURE DE COMPLEXITE DE PROGRAMMES ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER *
        **************************

(* gc *)
program t ;

var def , i , n : integer ;

begin
  for i := 1 to n do
    def := def + i
end .
```

```
        **********************************
        *  VISUALISATION DE L'ARBRE  *
        **********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :t
! ! ! !  ( int0_03: 36 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( inst_pour: 28 )
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! !  ( list_pour_1: 134 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
```

```
      **********************
      *  LES OPERATEURS  *
      **********************

Nombre total d'operateurs  :    6

Les operateurs :
        +                                       1
        :=                                      3
        begin                                   1
        for                                     1
        to                                      1


Nombre d'operateurs distincts :   5
```

****
* LES OPERANDES *
****

Nombre total d'operandes : 6

Les operandes :

i	1
def	2
i	2
n	1

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 2

L'intervalle Cycmin:N1 : 2 : 6

Longueur du programme : 1.960954e+01

Volume du programme : 3.303910e+01

Niveau du programme : 2.666667e-01

Le coefficient de Halstead : 1.426466e+02

****
****
**
**
**
**
**
**
**
**
**
**
**
****
****

MESURE DE COMPLEXITE DE PROGRAMMES
ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER    *
        **************************

(* gc *)
program t2 ;

var def , i , n : integer ;

begin
  i := 1 ;
  while i <= n do
    begin
      def := def + i ;
      i := i + 1
    end
end .
```

```
        **********************************
        *  VISUALISATION DE L'ARBRE  *
        **********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 2  VALUE :t2
! ! ! !  ( int0_03: 36 )
! ! ! ! !  ( inti_15: 33 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 103 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! !  ( inst_tant_que: 64 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! !  ( oper_15: 169 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 11

Les operateurs :

+	2
:=	3
;	2
<=	1
begin	2
while	1

Nombre d'operateurs distincts : 6

****
* LES OPERANDES *
****

Nombre total d'operandes : 10

Les operandes :

1	2
def	2
i	5
n	1

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 2

L'intervalle Cycmin:Ni : 2 : 11

Longueur du programme : 2.350978e+01

Volume du programme : 5.9760+9e+01

Niveau du programme : 1.333333e-01

Le coefficient de Halstead : 5.232037e+02

****
****
**
**
** MESURE DE COMPLEXITE DE PROGRAMMES **
** ECRITS EN PASCAL **
**
**
** memoire realise par GREGOIRE CH. **
**
** 85-86 **
**
****
****

```
        **************************
        * PROGRAMME A MESURER    *
        **************************

(* gc *)
program t2 ;

var def , i , n : integer ;

begin
  if i < n
    then begin
      def := def + 1 ;
      i := i + 1
    end
    else i := i + 1
end .
```

```
        **************************
        *  VISUALISATION DE L'ARBRE  *
        **************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 2  VALUE :t2
! ! ! !  ( int0_03: 36 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 11

Les operateurs :

+	3
:=	3
;	1
<	1
begin	2
if+else	1

Nombre d'operateurs distincts : 6

****
* LES OPERANDES *
****

Nombre total d'operandes : 11

Les operandes :

1	3
def	2
i	5
n	1

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 2

L'intervalle Cycmin:N1 : 2 : 11

Longueur du programme : 2.350978e+01

Volume du programme : 7.308242e+01

Niveau du programme : 1.212121e-01

Le coefficient de Halstead : 6.029299e+02

MESURE DE COMPLEXITE DE PROGRAMMES ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER *
        **************************

(* gc *)
program t2 ;

var def , i , n : integer ;

begin
  if i < n
    then def := def + 1 ;
  i := i + 1
end .
```

```
        **********************************
        *  VISUALISATION DE L'ARBRE  *
        **********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -135 )LENGTH : 2  VALUE :t2
! ! ! !  ( int0_03: 36 )
! ! ! ! !  ( int1_15: 33 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( if_then: 61 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :n
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 3  VALUE :def
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :i
! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :1
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 8

Les operateurs :

+	2
:=	2
;	1
<	1
begin	1
if	1

Nombre d'operateurs distincts : 6

****
* LES OPERANDES *
****

Nombre total d'operandes : 8

Les operandes :

i	2
def	2
i	3
n	1

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 2

L'intervalle Cycmin:Ni : 2 : 3

Longueur du programme : 2.350978e+01

Volume du programme : 5.315085e+01

Niveau du programme : 1.666667e-01

Le coefficient de Halstead : 3.189051e+02

MESURE DE COMPLEXITE DE PROGRAMMES ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-36

```
        **************************
        * PROGRAMME A MESURER *
        **************************

(* gc *)
program test ;

var x , j : integer ;

begin
  if x < j
    then x := x + 10
    else x := x - 5
end .
```

```
        **********************************
        *  VISUALISATION DE L'ARBRE  *
        **********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_03: 86 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 7

Les operateurs :

+	1
-	1
:=	2
<	1
begin	1
if+else	1

Nomore d'operateurs distincts : 6

****
* LES OPERANDES *
****

Nombre total d'operandes : 8

Les operandes :

10	1
5	1
j	1
x	5

Nombre d'operandes distincts : 4

```
**********************
*  LES COMPLEXITES   *
**********************
```

Le coefficient de McCabe : 2

L'intervalle Cycmin:N1 : 2 : 7

Longueur du programme : 2.350978e+01

Volume du programme : 4.982892e+01

Niveau du programme : 1.666667e-01

Le coefficient de Halstead : 2.989736e+02

MESURE DE COMPLEXITE DE PROGRAMMES ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER    *
        **************************

(* gc *)
program test ;

var x , j : integer ;

procedure comp ( j : integer ; var x : integer ) ;

begin
  if x < j
    then x := x + 10
    else x := x - 5
end ;

begin
  comp ( j , x )
end .
```

```
        **********************************
        *  VISUALISATION DE L'ARBRE  *
        **********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 76 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_01: 20 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_decl_prfo: 109 )
! ! ! ! ! !  ( fonct_proc: 10 )
! ! ! ! ! ! !  ( decl_proc_1: 54 )
! ! ! ! ! ! ! !  ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! !  ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! !  ( int0_04: 78 )
! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! !  ( ir_then_else: 27 )
! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! !  ( part_inst: 120 )
```

```
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( inst_appe: 128 )
! ! ! ! ! ! ! !  ( rec_appel: 59 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :como
! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 10

Les operateurs :

+	1
,	1
-	1
:=	2
<	1
begin	2
comp	1
if+else	1

Nombre d'operateurs distincts : 8

****
* LES OPERANDES *
****

Nombre total d'operandes : 10

Les operandes :

10	1
5	1
J	2
x	6

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 2

L'intervalle Cycmin:N1 : 2 : 10

Longueur du programme : 3.200000e+01

Volume du programme : 7.169925e+01

Niveau du programme : 1.000000e-01

Le coefficient de Halstead : 7.169925e+02

MESURE DE COMPLEXITE DE PROGRAMMES
ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER *
        **************************

(* gc *)
program test ;

var x , j : integer ;

begin
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5
end .
```

```
        ********************************
        *  VISUALISATION DE L'ARBRE    *
        ********************************

! ( annot : -1 )
! ! ( formalism : -1 ) go
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_03: 36 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 14

Les operateurs :

+	2
-	2
:=	4
;	1
<	2
begin	1
if+else	2

Nombre d'operateurs distincts : 7

****
* LES OPERANDES *
****

Nombre total d'operandes : 16

Les operandes :

10	2
6	2
j	2
x	10

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 3

L'intervalle Cycmin:Ni : 3 : 14

Longueur du programme : 2.766148e+01

Volume du programme : 1.037829e+02

Niveau du programme : 7.142857e-02

Le coefficient de Halstead : 1.452961e+03

MESURE DE COMPLEXITE DE PROGRAMMES
ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER    *
        **************************

(* gc *)
program test ;

var x , j : integer ;

procedure comp ( j : integer ; var x : integer ) ;

begin
  if x < j
    then x := x + 10
    else x := x - j
end ;

begin
  comp ( j , x ) ;
  comp ( j , x )
end .
```

```
        **********************************
        *  VISUALISATION DE L'ARBRE  *
        **********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_01: 20 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 103 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_decl_prfo: 109 )
! ! ! ! ! !  ( fonct_proc: 10 )
! ! ! ! ! ! !  ( decl_proc_1: 54 )
! ! ! ! ! ! ! !  ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! !  ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! !  ( int0_04: 78 )
! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! !  ( part_inst: 120 )
```

```
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( inst_appe: 125 )
! ! ! ! ! ! ! !  ( rec_appel: 59 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! !  ( inst_appe: 125 )
! ! ! ! ! ! ! !  ( rec_appel: 59 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 13

Les operateurs :

+	1
,	2
-	1
:=	2
;	1
<	1
begin	2
comp	2
if+else	1

Nombre d'operateurs distincts : 9

****
* LES OPERANDES *
****

Nombre total d'operandes : 12

Les operandes :

10	1
5	1
J	3
x	7

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 2

L'intervalle Cycmin:N1 : 2 : 13

Longueur du programme : 3.652933e+01

Volume du programme : 9.251099e+01

Niveau du programme : 7.407407e-02

Le coefficient de Halstead : 1.248898e+03

MESURE DE COMPLEXITE DE PROGRAMMES ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

35-86

```
        **************************
        * PROGRAMME A MESURER *
        **************************

(* gc *)
program test ;

var x , j : integer ;

begin
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5
end .
```

```
        ********************************
        *  VISUALISATION DE L'ARBRE    *
        ********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_33: 36 )
! ! ! ! !  ( inti_15: 33 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 103 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
```

```
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 21

Les operateurs :

+	3
-	3
:=	6
;	2
<	3
begin	1
if+else	3

Nombre d'operateurs distincts : 7

****
* LES OPERANDES *
****

Nombre total d'operandes : 24

Les operandes :

10	3
5	3
j	3
x	15

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 4

L'intervalle Cycmin:N1 : 4 : 21

Longueur du programme : 2.765148e+01

Volume du programme : 1.556744e+02

Niveau du programme : 4.761905e-02

Le coefficient de Halstead : 3.269163e+03

MESURE DE COMPLEXITE DE PROGRAMMES ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER    *
        **************************

(* gc *)
program test ;

var x , j : integer ;

procedure comp ( j : integer ; var x : integer ) ;

begin
  if x < j
    then x := x + 10
    else x := x - 5
end ;

begin
  comp ( j , x ) ;
  comp ( j , x ) ;
  comp ( j , x )
end .
```

```
        ********************************
        *  VISUALISATION DE L'ARBRE    *
        ********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_01: 20 )
! ! ! ! !  ( int1_15: 33 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_decl_prfo: 109 )
! ! ! ! ! !  ( fonct_proc: 10 )
! ! ! ! ! ! !  ( decl_proc_1: 54 )
! ! ! ! ! ! ! !  ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! !  ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! !  ( int0_04: 78 )
! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! !  ( part_inst: 120 )
```

```
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( inst_appe: 125 )
! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
```

```
**********************
* LES OPERATEURS *
**********************
```

Nombre total d'operateurs : 16

Les operateurs :

+	1
,	3
-	1
:=	2
;	2
<	1
begin	2
comp	3
if+else	1

Nombre d'operateurs distincts : 9

****
* LES OPERANDES *
****

Nombre total d'operandes : 14

Les operandes :

10	1
5	1
j	4
x	3

Nombre d'operandes distincts : 4

```
**********************
*  LES COMPLEXITES   *
**********************
```

Le coefficient de McCabe : 2

L'intervalle Cycmin:N1 : 2 : 16

Longueur du programme : 3.651953e+01

Volume du programme : 1.110132e+02

Niveau du programme : 6.349206e-02

Le coefficient de Halstead : 1.748458e+03

MESURE DE COMPLEXITE DE PROGRAMMES ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER *
        **************************

(* gc *)
program test ;

var x , j : integer ;

begin
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5
end .
```

```
        **********************************
        *  VISUALISATION DE L'ARBRE  *
        **********************************

! ( annot : -1 )
! ! ( formalism : -1 ) go
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -135 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_03: 36 )
! ! ! ! !  ( int1_15: 33 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_exor: 29 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
```

```
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -135 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
```

```
**********************
* LES OPERATEURS *
**********************
```

Nombre total d'operateurs : 28

Les operateurs :

+	4
-	4
:=	3
;	3
<	4
begin	1
if+else	4

Nombre d'operateurs distincts : 7

****
* LES OPERANDES *
****

Nombre total d'operandes : 32

Les operandes :

10	4
5	4
J	4
x	20

Nombre d'operandes distincts : 4

****
* Les COMPLEXITES *
****

Le coefficient de McCabe : 5

L'intervalle Cycmin:N1 : 5 : 28

Longueur du programme : 2.765148e+01

Volume du programme : 2.073659e+02

Niveau du programme : 3.571429e-02

Le coefficient de Halstead : 5.811845e+03

****
****
**
**
**
**
**
**
**
**
**
**
**
****
****

MESURE DE COMPLEXITE DE PROGRAMMES
ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER    *
        **************************

(* gc *)
program test ;

var x , j : integer ;

procedure comp ( j : integer ; var x : integer ) ;

begin
  if x < j
    then x := x + 10
    else x := x - 5
end ;

begin
  comp ( j , x ) ;
  comp ( j , x ) ;
  comp ( j , x ) ;
  comp ( j , x )
end .
```

```
        ********************************
        *  VISUALISATION DE L'ARBRE    *
        ********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_01: 20 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_decl_prfo: 109 )
! ! ! ! ! !  ( fonct_proc: 10 )
! ! ! ! ! ! !  ( decl_proc_1: 54 )
! ! ! ! ! ! ! !  ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! !  ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! !  ( int0_04: 78 )
! ! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! !  ( part_inst: 120 )
```

```
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 19

Les operateurs :

+	1
,	4
-	1
:=	2
;	3
<	1
begin	2
comp	4
if+else	1

Nombre d'operateurs distincts : 9

****
* LES OPERANDES *
****

Nombre total d'operandes : 16

Les operandes :

10	1
5	1
J	5
x	9

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 2

L'intervalle Cycmin:N1 : 2 : 19

Longueur du programme : 3.652933e+01

Volume du programme : 1.295154e+02

Niveau du programme : 5.555556e-02

Le coefficient de Halstead : 2.331277e+03

MESURE DE COMPLEXITE DE PROGRAMMES
ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **************************
        * PROGRAMME A MESURER *
        **************************

(* gc *)
program test ;

var x , j : integer ;

begin
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5 ;
  if x < j
    then x := x + 10
    else x := x - 5
end .
```

```
        **********************************
        *  VISUALISATION DE L'ARBRE  *
        **********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_03: 36 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 103 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
```

```
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
```

****
* LES OPERATEURS *
****

Nombre total d'operateurs : 35

Les operateurs :

+	5
-	5
:=	10
;	4
<	5
begin	1
if+else	5

Nombre d'operateurs distincts : 7

****
* LES OPERANDES *
****

Nombre total d'operandes : 40

Les operandes :

10	5
5	5
j	5
x	25

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 6

L'intervalle Cycmin:Ni : 6 : 35

Longueur du programme : 2.765148e+01

Volume du programme : 2.594574e+02

Niveau du programme : 2.857143e-02

Le coefficient de Halstead : 9.081003e+03

MESURE DE COMPLEXITE DE PROGRAMMES
ECRITS EN PASCAL

memoire realise par GREGOIRE CH.

85-86

```
        **********************
        * PROGRAMME A MESURER *
        **********************

(* gc *)
program test ;

var x , j : integer ;

procedure comp ( j : integer ; var x : integer ) ;

begin
  if x < j
    then x := x + 10
    else x := x - 5
end ;

begin
  comp ( j , x ) ;
  comp ( j , x ) ;
  comp ( j , x ) ;
  comp ( j , x ) ;
  comp ( j , x )
end .
```

```
        ********************************
        *  VISUALISATION DE L'ARBRE    *
        ********************************

! ( annot : -1 )
! ! ( formalism : -1 ) gc
! !  ( point_d_entree: 75 )
! ! !  ( program: 34 )
! ! ! !  ( tete_prog: 76 )
! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :test
! ! ! !  ( int0_01: 20 )
! ! ! ! !  ( int1_15: 83 )
! ! ! ! ! !  ( decl_var: 107 )
! ! ! ! ! ! !  ( part_decl_var: 108 )
! ! ! ! ! ! ! !  ( decl_var_1: 9 )
! ! ! ! ! ! ! ! !  ( decl_var_2: 53 )
! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! !  ( part_decl_prfo: 109 )
! ! ! ! ! !  ( fonct_proc: 10 )
! ! ! ! ! ! !  ( decl_proc_1: 54 )
! ! ! ! ! ! ! !  ( en_tete_proc_2: 55 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_list_2: 11 )
! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! ! !  ( var_group: 115 )
! ! ! ! ! ! ! ! ! ! !  ( group_para: 56 )
! ! ! ! ! ! ! ! ! ! ! !  ( suit_ident: 4 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 7  VALUE :integer
! ! ! ! ! ! ! ! !  ( int0_04: 78 )
! ! ! ! ! ! ! ! !  ( part_inst: 120 )
! ! ! ! ! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! ! ! ! ! !  ( if_then_else: 27 )
! ! ! ! ! ! ! ! ! ! ! !  ( test_expr: 29 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_13: 167 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_21: 172 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 2  VALUE :10
! ! ! ! ! ! ! ! ! ! ! !  ( inst_affec: 60 )
! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! !  ( expr_oper: 30 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( oper_22: 173 )
! ! ! ! ! ! ! ! ! ! ! ! ! !  ( etiq: -181 )LENGTH : 1  VALUE :5
! ! ! ! !  ( part_inst: 120 )
```

```
! ! ! ! ! !  ( list_inst: 12 )
! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
! ! ! ! ! ! ! !  ( inst_appe: 126 )
! ! ! ! ! ! ! ! !  ( rec_appel: 69 )
! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 4  VALUE :comp
! ! ! ! ! ! ! ! ! !  ( list_para_effe: 17 )
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :j
! ! ! ! ! ! ! ! ! ! !  ( ident: -185 )LENGTH : 1  VALUE :x
```

```
**********************
*  LES OPERATEURS    *
**********************
```

Nombre total d'operateurs : 22

Les operateurs :

+	1
,	5
-	1
:=	2
;	4
<	1
begin	2
comp	5
if+else	1

Nombre d'operateurs distincts : 9

****
* LES OPERANDES *
****

Nombre total d'operandes : 18

Les operandes :

10	1
5	1
J	6
x	10

Nombre d'operandes distincts : 4

****
* LES COMPLEXITES *
****

Le coefficient de McCabe : 2

L'intervalle Cycmin:N1 : 2 : 22

Longueur du programme : 3.652933e+01

Volume du programme : 1.480176e+02

Niveau du programme : 4.938272e-02

Le coefficient de Halstead : 2.997356e+03